BSD MAGAZINE

FOR NOVICE AND ADVANCED USERS

FREEBSD ON RAILS
CREATE YOUR RUBY ON RAILS APP

VOL.7 NO.6
ISSUE 06/2013(47)
1898-9144

iXsystems®

855-GREP-4-IX
www.iXsystems.com
Enterprise Servers and Storage for Open Source

✓ Rock-Solid Performance

✓ Professional In-House Support

E5-2600

High Performance,
High Density Servers for
Data Center, Virtualization, & HPC

MODEL: iXR-22X4IB

http://www.iXsystems.com/e5

768GB
of RAM in 1U

KEY FEATURES

iXR-22X4IB	iXR-1204+10G
• Dual Intel® Xeon® Processors E5-2600 Family per node • Intel® C600 series chipset • Four server nodes in 2U of rack space • Up to 256GB main memory per server node • One Mellanox® ConnectX QDR 40Gbp/s Infiniband w/QSFP Connector per node • 12 SAS/SATA drive bays, 3 per node • Hardware RAID via LSI2108 controller • Shared 1620W redundant high-efficiency Platinum level (91%+) power supplies	• Dual Intel® Xeon® Processors E5-2600 Family • Intel® C600 series chipset • Intel® X540 Dual-Port 10 Gigabit Ethernet Controllers • Up to 16 Cores and 32 process threads • Up to 768GB main memory • Four SAS/SATA drive bays • Onboard SATA RAID 0, 1, 5, and 10 • 700W high-efficiency redundant power supply with FC and PMBus (80%+ Gold Certified)

Call iXsystems toll free or visit our website today! **1-855-GREP-4-IX | www.iXsystems.com**

High-Density iXsystems Servers powered by the Intel® Xeon® Processor E5-2600 Family and Intel® C600 series chipset can pack up to 768GB of RAM into 1U of rack space or up to 8 processors - with up to 128 threads - in 2U.

On-board 10 Gigabit Ethernet and Infiniband for Greater Throughput in less Rack Space.

Servers from iXsystems based on the Intel® Xeon® Processor E5-2600 Family feature high-throughput connections on the motherboard, saving critical expansion space. The Intel® C600 Series chipset supports up to 384GB of RAM per processor, allowing performance in a single server to reach new heights. This ensures that you're not paying for more than you need to achieve the performance you want.

The iXR-1204 +10G features dual onboard 10GigE + dual onboard 1GigE network controllers, up to 768GB of RAM and dual Intel® Xeon® Processors E5-2600 Family, freeing up critical expansion card space for application-specific hardware. The uncompromised performance and flexibility of the iXR-1204 +10G makes it suitable for clustering, high-traffic webservers, virtualization, and cloud computing applications - anywhere you need the most resources available.

For even greater performance density, the iXR-22X4IB squeezes four server nodes into two units of rack space, each with dual Intel® Xeon® Processors E5-2600 Family, up to 256GB of RAM, and an on-board Mellanox® ConnectX QDR 40Gbp/s Infiniband w/QSFP Connector. The iXR-22X4IB is perfect for high-powered computing, virtualization, or business intelligence applications that require the computing power of the Intel® Xeon® Processor E5-2600 Family and the high throughput of Infiniband.

HIGH Throughput & INCREDIBLE Performance Density

IXR-1204+10G: **10GbE On-Board**

4 Server Nodes in 2U

IXR-22X4IB

iXsystems®

intel® inside™ Xeon®

Call iXsystems toll free or visit our website today! **1-855-GREP-4-IX | www.iXsystems.com**

Dear Readers,

The June issue of BSD Magazine is dedicated to the Ruby scripting language with the use of Ruby Version Manager (RVM) and Ruby on Rails (RoR), the Ruby's web framework. Moreover, on the following pages, you will find articles about Sofin software installer and security updates for OpenBSD packages.

We start with Rob's column, where he will discuss how BBC has abandoned a $150m IT project and suspended the CTO responsible.

In the What's New section, Daniel Dettlaff announces the first release of Sofin, a software installer that provides a new way to build software these days. This tool will eliminate endless problems with software requirements, user demands, and that entire mess.

Next, we will show how to back up a server step by step on a regular basis to prevent the loss of data.

Then you will have a chance to test the Ruby on Rails framework on FreeBSD. The RoR is a framework that is very well-known in the world of web-development. It allows you to create fully featured websites semi-automatically.

This month's Dev Corner covers the Push Button Installer (PBI) format which is an easy-to-use package format for end-user applications. It covers EasyPBI as well. EasyPBI is a tool designed to simplify the generation of these PBI packages.

In the Admin section, Thibaut Deloffre explores the Ruby Version Manager, which is a great tool to manage several Ruby binaries without dependency breaks.

Then in the Extras section, Petr Topiarz will talk a bit about binpatches, -stable package updates and will show how to start using the update service on OpenBSD.

Finally, Egoitz Aurrekoetxea Aurre will list Xen Cloud Platform's advantages and will show how to take them with FreeBSD. Moreover, he will demonstrate how to run FreeBSD in XCP.

We hope you will enjoy this issue and find many interesting articles!

Kamil Sobieraj
Editor of BSD Magazine
& BSD Team

MAGAZINE BSD

Editor in Chief:
Ewa Dudzic
ewa.dudzic@software.com.pl

Supportive Editor:
Kamil Sobieraj
kamil.sobieraj@software.com.pl

Contributing:
Antonio Francesco Gentile, Daniel (Dmilith) Dettlaff, Rob Somerville, Julien Grillot, Ken Moore, Thibaut Deloffre, Petr Topiarz, Egoitz Aurrekoetxea Aurre

Top Betatesters & Proofreaders:
Luca Ferrari, Imad Soltani, Eric Geissinger, Eduardo Morras, Scott Sealy, Denise Ebery, Dru Lavigne, Annie A. Zhang

Special Thanks:
Denise Ebery
Luca Ferrari

Art Director:
Ireneusz Pogroszewski

DTP:
Ireneusz Pogroszewski
ireneusz.pogroszewski@software.com.pl

Senior Consultant/Publisher:
Paweł Marciniak
pawel@software.com.pl

CEO:
Ewa Dudzic
ewa.dudzic@software.com.pl

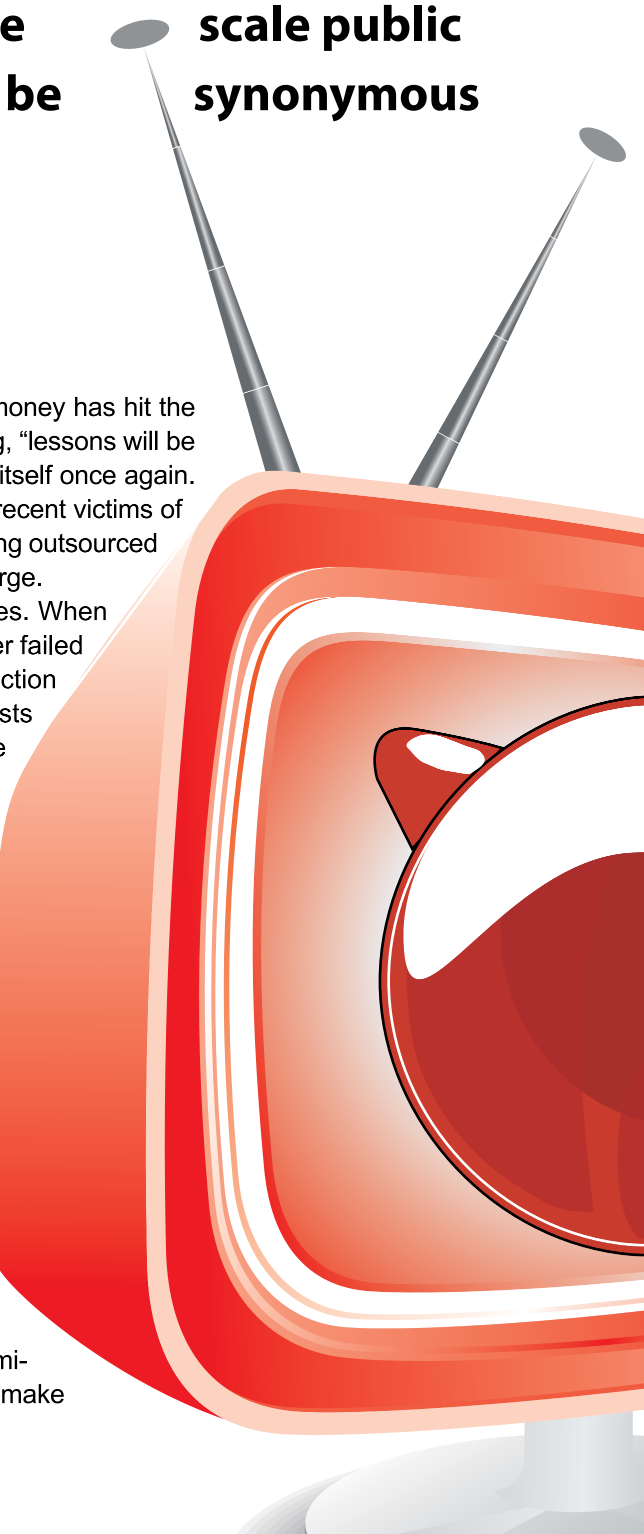
Production Director:
Andrzej Kuca
andrzej.kuca@software.com.pl

Advertising Sales:
Kamil Sobieraj
kamil.sobieraj@software.com.pl

Publisher:
Software Media Sp. z o.o. SK
ul. Bokserska 1, 02-682 Warszawa
Poland
worldwide publishing
tel: 1 917 338 36 31
www.bsdmag.org

The British Broadcasting Corporation (BBC)

has abandoned a $150m IT project and suspended the CTO responsible. What is it about large scale public sector IT projects that causes them to be synonymous with failure?

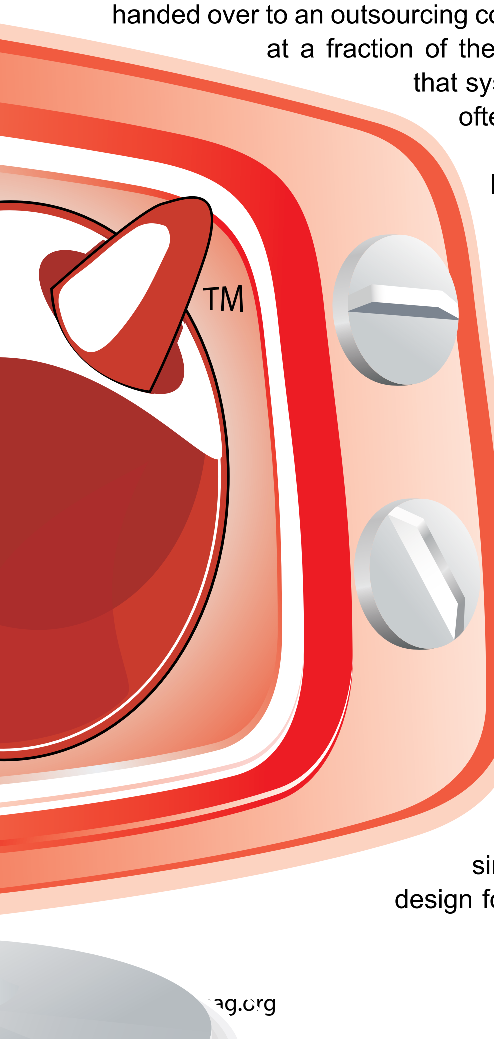


Here we go again. Another large IT project funded by taxpayers' money has hit the wall. Of course there will be an investigation, much finger pointing, "lessons will be learned", and further down the road, history will no doubt repeat itself once again. The defence, health, employment, and nuclear sectors have been other recent victims of failure, but it seems that the message is not getting through – whether using outsourced or in-house resources, the spectre of catastrophic project failure looms large.

If we look at other sectors, we very rarely hear of major project failures. When was the last time it was announced that a major automotive manufacturer failed to build a new plant or an aircraft manufacturer abandoned the production of a new aircraft? Of course, there is always the financial problem – costs escalate, and the project is no longer viable. How many times have we heard of costs doubling, tripling and more – yet it all depends on how vital it is to complete the project. Sometimes the plug is pulled (as in this case). In other cases, the project carries on until something is delivered. Comparing the building and IT sectors, construction projects seem to run almost flawlessly, yet the models used (Strong project management, tight budget controls, using specialist contractors etc.) don't always seem to translate into the IT environment. At first glance, this seems illogical – both sectors are engineering based, the science is well understood, and there are lots of examples of good practice to use as a template. So what is it that plagues the IT industry with so many public and embarrassing failures?

The first problem is that we are dealing with highly complex systems. System A may be very stable, reliable, but slow. It doesn't scale. It was designed and built quite possibly in an age before the Internet was conceived. Yet it is the bedrock of the enterprise, everybody knows that it works, but at some point, additional functionality and expansion are required. It would be too costly to take it out and start again. The amount of downtime to the organisation would be prohibitive, and migrating the system to another platform would be financially prohibitive and make the project too risky. So over the years, compromises have been reached, bits have been bolted on (quite possibly undocumented) and the engineer responsible is long gone. The system has evolved way past its original specification and morphed systemically into something very different from what was originally commissioned. Quite possibly, due to organisational or social culture and the refusal to accept that knowledge is a priceless asset, there is a shortage of experts on that system. This became clear when fixing Year 2000 issues. COBOL for years was considered a dead language. Then suddenly, enterprises realised they needed expertise that had been squeezed out of the market by lack of demand. Suddenly, if you were a COBOL expert, you could just about name your price.

There is also a degree of reticence about sharing and documenting knowledge that will decrease your value in the marketplace. Once again, the thorny issue of Intellectual Property raises its ugly head. Where do you ethically draw the line between personal innovation and the property of your employer? Most engineers I know thrive on solving problems and continually want to improve systems and make them better for their users. This clashes though with the commercial reality where employers want a complete "knowledge dump" and then expect this to be handed over to an outsourcing company to be supported at a fraction of the cost. Is it any wonder that systems documentation is often of such poor quality?



Then there are the political and commercial drivers behind the system. The old adage goes "You can have it cheaply, quickly, or properly. Pick 2". Too often, compromises are made at the early stage of the project that have a major impact on either how long it will take (missed deadlines) or on the amount of resources required to accomplish X, Y, or Z (Cost overruns). The rules are really quite straightforward: keep it as simple as possible and design for extensibility and flexibility, but make sure the foundation is strong and has sufficient redundancy to accommodate unexpected changes in the future. In other words, you need to hit that sweet spot between an under- and over-engineered system. The essential thinking is this – assume the project targets are subject to change without notice and pick the technology base that gives you maximum flexibility, so that you will not have to start from scratch if the game changes dramatically.

Finally, there is the inherent disconnect. Every project has a few, to some degree or another, and this is probably the key reason why the BBC project failed so miserably. A minor disconnect will come back and haunt you for years, but it will not have a major impact on the overall viability of the project. Provided you are aware and don't try and build on it, the project should be a success. A good example of this is where a section of the project is contracted out and the supplier delivers just enough to tick all the boxes, but everybody knows the code and inherent design is poor and getting the supplier to engage is hard work. Legally and contractually, they can walk away, but everybody knows in their heart of hearts that in a year or two, that work will have to be redone from scratch. With good fortune, the system itself will carry on until end of life without any major impact. A major disconnect, on the other hand, will cause the project to break. Inevitably, this comes from disregarding the "cheaply, quickly, properly" rule, or to use more formal project management language, the triple constraint model. For whatever reason, it is decided that the project can support three rather than two immutable deliverables, and then havoc reigns. The tragedy is that this will have been raised at some point, but there are none so deaf as those that wish not to hear. As often in large organisations, the culture prevents open and honest communication, and it takes a brave individual to swim against the tide and deliver bad news or resist the official line. Worse still, rather than having a strong unified focus and leadership, we have fragmentation with committees, different contesting philosophies, and the attitude that "it is the other guy's problem". The elephant finally enters the room – never to depart.

ROB SOMERVILLE

Rob Somerville has been passionate about technology since his early teens. A keen advocate of open systems since the mid eighties, he has worked in many corporate sectors including finance, automotive, airlines, government and media in a variety of roles from technical support, system administrator, developer, systems integrator and IT manager. He has moved on from CP/M and nixie tubes but keeps a soldering iron handy just in case.

Sofin, the Software Installer

If you've ever tried building software for your server without getting mad and frustrated, without approaching endless problems with software requirements, user demands and that entire mess, you should probably know that there's a solution available that addresses these problems. It's called Sofin.

What you will learn…

- How software is built nowadays in the Open Source world.
- Why you should avoid the way of building software that's currently considered the standard, and look at something that's designed better.
- How Sofin cures your headaches – the details not mentioned in the project README file.

What you should know…

- You should know how to build software from source.
- You should know what a shared library is and have some basic understanding of how compiler and linker work.
- If you want to make your own software definitions, you should have (at least) basic knowledge on shell scripting.

Basically, every *NIX compliant server system that's currently used has its own way of approaching software. I'll explain it by example:

Let's assume you have a clean installation of your favorite OS base. It's FreeBSD 9.1 in my case – that's my major production server platform of choice. There's `/usr/bin`, `/usr/sbin` and `/bin`, where all base system software executables reside. In short – every piece of software is put into a "common bag" of binaries (`bin`, `sbin`) and libraries (`lib`, `lib32`, `lib64`). If you want to use software which isn't provided by your base system, say Ruby 2.0, you'll need to install it manually. I'll skip the part about installing software from prebuilt binary packages, mostly because you won't find my example software in binary builds, and you won't find binary builds for "your software", especially if you're dealing with custom or old server configurations.

So you end up building it from source manually or through ports. Each additional software built from source will go into yet another "common bag" in `/usr/local/` by default.

So, what's wrong with this approach?

The thing is – when you're creating a server, you usually want it to be used by your users, right? A user is an unprivileged entity that only wants to run some software.

Users demand that you build reliable software for them to use. But you can't give them that with an FHS approach to software. I'll explain by another real life example: Let's assume that, after Ruby, you've also installed PostgreSQL, MySQL, Redis, Imagemagick, Cairo and a few other pieces of software in your `/usr/local/` bag. You end up with tons of common libraries (that you usually know nothing about), all put in just one place.

It should be fine, right? Not even close. Try to uninstall some of them now. No `make uninstall` available. What now?

But the real fun begins when you want to do a security update for one of your libraries that's commonly used (and shared) by some software. How many times did you do an upgrade of ports binaries/libraries and then end up with a part of the software broken? (For *BSD there's an UPDATING file in ports with information about how to do software updates, but usually it gives you nothing, and the problem remains, which, in short, depends on which ports you installed, in what order, and so on).

I had enough after a couple of times of reinstalling all my software because of one library change. My system became a mess and I lost control of my software and their dependencies. But this is only one side of the coin. There's more: for example, what will you do if you have two applications that require different versions of the same library to be linked with?

The solution is to do some kind of a hack – usually by building a prefixed library, prefixed binaries, or by giving options manually to a build script and so on. After a couple hacks of that kind, you still end up with a mess on your production machine which – I assume – is simply undesired.

Possibly the worst of all – you need to remember what did you do to make it work. There's also the third side of the coin, user privileges.

Ruby is a great example of such an issue. If your users want to install their gems (doesn't matter if they use Bundler or not), they'll require root privileges to write their gems into default `/usr/local/lib/directory`. Those gems will be common for all users but not their own. This is where all the hacky solutions like rbenvand rvm are "shining?" Not at all. They're just ugly hacks, created on top of bad software architecture. Believe it or not, these problems are just the tip of the iceberg.

How Sofin was born

Some say that the best software is born thanks to a developer's rage. Sofin was one of those projects that started spontaneously after I just gave up hacking one of my servers.

I wanted my software to be reliable, without shared dependencies, bundled, owned by user, yet fully customizable. I wanted it to work on all POSIX-compliant systems and to be designed with simplicity in mind (KISS rule). I also wanted it to be BSD-licensed because I've had enough fighting with GPL/GNU stuff. And well… Sofin was born.

Sofin celebrated its second birthday in May 2013. Currently, there are more than 200 definitions of server software available. You may think that 200 is nothing when compared to 20,000 ports. Yes, but how many of these ports definitions actually work? And how many of these are just X11 utilities? Which are just as dead, obsolete, or simply broken, and which aren't maintained anymore?

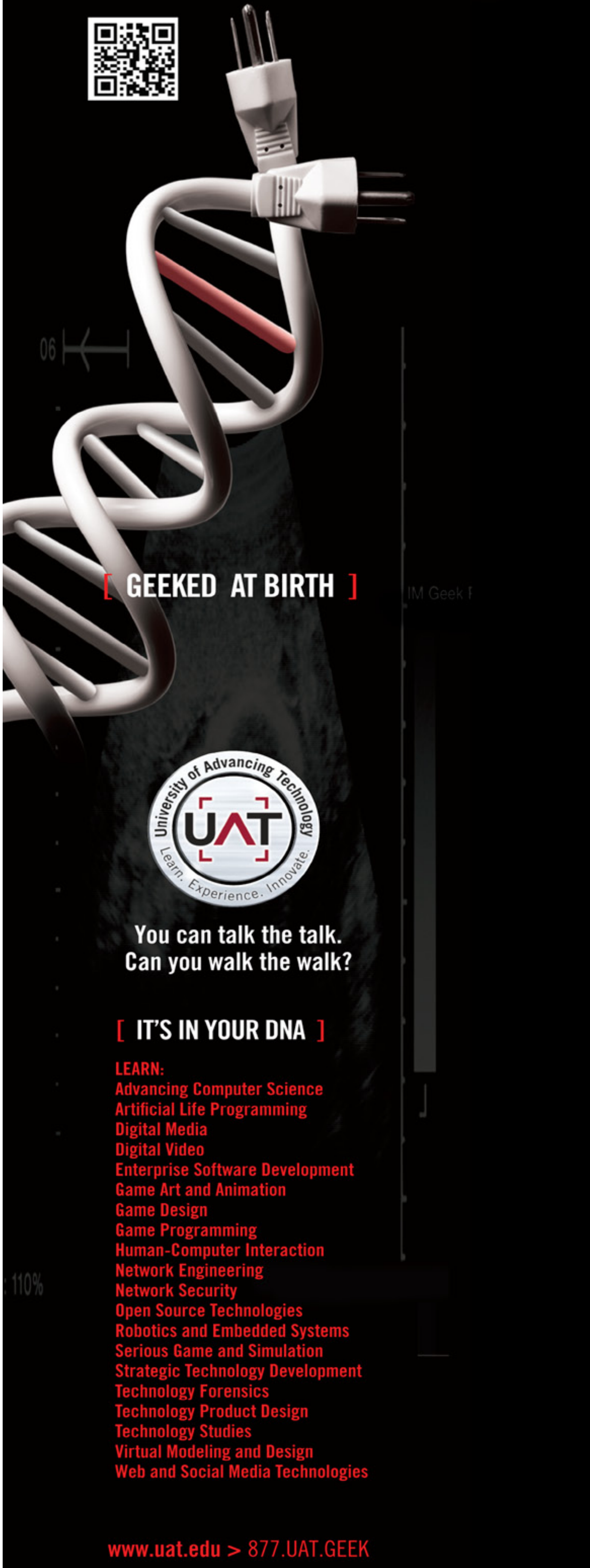
All Sofin software definitions are tested and used on FreeBSD 9.1, Debian 6.0, and Mac OS X 10.8. There's a policy that a definition isn't accepted in Sofin's repository unless it builds and installs correctly on all supported systems.

Sofin in depth

Differences from FHS standard in real life

In the late 90's, we had small disks. I used to work on a machine with 840 MB of disk space. This was probably the major reason for the FHS rule about keeping software in common prefixes: `/usr` and `/usr/local` – simply to save space. Each software depending on library X could just link to it in one common place. It was sufficient for simple software, simple solutions.

But the world is moving forward. Today, I have at least 1 TB of disk space on each server that runs software with tons of features and dependencies.


[GEEKED AT BIRTH]
University of Advancing Technology
UAT
Learn. Experience. Innovate.
You can talk the talk.
Can you walk the walk?
[IT'S IN YOUR DNA]
LEARN:
Advancing Computer Science
Artificial Life Programming
Digital Media
Digital Video
Enterprise Software Development
Game Art and Animation
Game Design
Game Programming
Human-Computer Interaction
Network Engineering
Network Security
Open Source Technologies
Robotics and Embedded Systems
Serious Game and Simulation
Strategic Technology Development
Technology Forensics
Technology Product Design
Technology Studies
Virtual Modeling and Design
Web and Social Media Technologies
www.uat.edu > 877.UAT.GEEK

The main idea of software bundling in Sofin was to get rid of that system-wide "shared nature" of binaries and libraries. I wanted to stop using `/usr/local` (Sofin will warn when this folder will exist on the production server) and never touch base system files in `/usr`. Each Sofin package has its own "root" directory (similar to that in `/usr`). By default, it's `~/Apps/YourApp/` where all software, libraries, and dependencies reside.

Here's why I mentioned disk space in the first place: each piece of software has its own copy of dependencies, hence they use more disk space. Usually it's up to 3 times more space than standard software. Not an issue these days, right? For some people who are more familiar with the BSD systems family, a software bundle might look similar to PBI packages from PC-BSD, but PBI bundles are far from simple. They're too complicated to define something as simple as software bundle.

The second difference is that PBIs aren't designed to be server software at all, they're just an imitation of an`*.app` bundle used by Apple. One more difference from the FHS approach is an additional `exports` directory in the root of each Sofin bundle. It adheres to the closed dependency model and is designed to provide access only to binaries that a user requires. The very important thing to know is that Sofin's shell setup won't ever set default `$PATH` access to `~/Apps/YourApp/bin/` nor `~/Apps/YourApp/sbin/`, but only to `~/Apps/YourApp/exports/`. If you want to have immediate access to a command from YourApp bundle, you'll need to add it to `APP_EXPORTS` in its definition, or manually run `sofin export your-command yourapp` after installation. That's it. Let's take a closer look at some more features.

Design assumptions

Sofin is written in probably the most primitive of all shell languages, the legacy sh. The surprising fact is that this simple language is probably one of the most powerful utilities to write software like Sofin. All configuration and every definition is also written in `sh`, hence you have the built-in scripting language into definitions for free. It gives you something that's very important: flexibility.

Here comes one of Sofin's major features – system integration with minimal interaction. By default, Sofin works in two modes: for user or for super user. When building software for regular users, it's put into `~/Apps/YourApp/`. When building it as a super user, it goes into `/Software/YourApp`. For a regular user, `~/.profile` file is generated after each software installation/uninstallation. For a super user, Sofin modifies `/etc/zshenv`, `/etc/bashrc` and `/etc/profile` files once (when installing Sofin) to support any `sh`-compliant shells, and then regenerates `/etc/profile_sofin` (the equivalent of user's `~/.profile`).

The idea behind `/Software` directory was to give the ability to install "base system extension software" without interacting with `/usr/bin` and `/usr/lib`. By default, software from `/Software` directory is common for all users (for example `Ccache`, `Clang`, `Git` and `Zsh` are recommended to be installed in `/Software`).

When building software as user, all software belongs to that particular user. It is very important that nobody in the system has access to your applications directory, and after a build, you can freely copy the entire app bundle to another machine and it will just work. The requirements of each software bundle comes from its own bundle and/or base system. No external dependencies are allowed. (At the time of writing this article, I'm working on binary builds for Sofin which will give the ability to skip the software build process, with drastic time savings and the additional ability to move software between users).

The next major feature is flat dependencies. Each software definition has its own, optional list of dependencies that will be installed in a given order before the very software. Sofin automatically detects library dependencies and builds destination software with them.

I won't mention all the Sofin features. If you're interested in all of them in detail, I already mentioned them on the project page at GitHub.

Installation and deployment: how Sofin affects environment

Installing Sofin in a new system might be considered a non-straightforward task. You start with installing Sofin itself (using detailed information obtained from the project page). Remember that the Sofin installation process is slightly different for each system family. The thing I didn't mention in the installation process on the project page is the deployment process of Sofin itself. After the installation, it's very important to do `sofin install base` as super user. It will install the base software like `Clang` and `Ccache` which is widely used by Sofin to build software later. If Sofin won't find `/Software/Clang/exports/clang` (and clang++), it will require gcc (and g++) installed in the system. Please consider gcc/ g++ slow and faulty – try avoiding these! Depending on your system, GNU compiler might fail to build certain software, and it's not widely tested (note that there are definitions, with defined requirement on gcc, hence it's still required to be installed). By default, Sofin will try to use `Clang` to build each software. If it also finds `Ccache`, it will give you a speed boost when you're compiling similar software dependencies between installed software. It also supports parallel builds by default (the amount of parallel tasks is equal to the amount of CPU cores available on your server). If the given software doesn't support one of the default approaches, they might be turned off explicitly in the definition file.

The next important issue is a shell implementation itself. Currently, the best supported shell is `zsh` (it's installed with "base" by default), so it's recommended to use a shell that reads standard shell initialization scripts on launch (`/etc/zshenv` for `zsh`). If something is not working, it's usually caused by improper shell configuration or conflicts caused by some third party software. Sofin demands (yes, it's not a policy but a requirement) sysadmins to have a clean system/ shell/ software configuration.

The next thing, which is very important for new users, is to invoke sofin reload (once every first user login).

It will (re)generate the `~/.profile` file and cause the current shell to reload it automatically. After that, you're free to install any defined software. After each installation, a shell reload is done automatically for the current shell, and the software is available to run immediately. If you're running multiple shell environments at once, you may require a sofin reload after the installation of new software.

Available definition features

Software definitions are based on a default definition in "defaults.def". This file contains all available settings that might be set for each definition. I'll mention only some of the ones that may not be as self-explanatory as the rest:

- `APP_NAME` – name of the software. It's a special value, used to name the software bundle directory.
- `APP_HTTP_PATH` – the address of the definition source archive.
- `FORCE_GNU_COMPILER` – an option to tell Sofin that the given definition can't be built using the Clang compiler.
- `APP_NO_CCACHE` – set to anything but "", if your software doesn't support building with Ccache.
- `DISABLE_ON` – an option that's required for software that isn't working on some systems. It's a space-separated list of system names (uname) on which the definition build will be skipped.
- `APP_EXPORTS` – a space-separated executables list taken from bundle bin/, sbin/ and libexec/ directories. Defines binaries to be exported for the given software bundle.
- `APP_REQUIREMENTS` – a space-separated list of definition names (without .def extension) to be installed, before the given definition itself (defines software dependencies).
- `APP_AFTER_*_CALLBACK` – callbacks invoked after given stages of the build, where '*' is a stage name. Current stage names (in order of execution): `UNPACK`, `CONFIGURE`, `MAKE`, `INSTALL`, `PATCH`, `EXPORT`. Callback might contain shell commands or sh function name (this function must be defined in the definition itself), which will be called by name. For an example, look on sbt.def in available definitions.
- `APP_SHA` – SHA1 checksum of the source archive of the given definition. If the check fails, Sofin will assume a truncated/broken file, and retry a download from the software source server.
- `APP_CURRENT_VERSION` – used to determine the availability of a new software version (usually from the software home page). Look into ruby.def definition for an example.
- `APP_CONFLICTS_WITH` – a space-separated list of Bundle names (capitalized), that will export the same binaries as a given definition (under the hood – it just renames exports/ to exports-disabled/ for each conflicting definition).
- `REQUIRE_ROOT_ACCESS` – set for definitions that must be built as root (for example, Openafs which includes kernel module).

Sofin in action

Sofin supports two "kinds" of definitions. The first is a regular definition (*.def files – more details in README file in the git repository) which has all the information required to build the given software. The second is a definitions list, which is just a simple text file with newline separated names (again without extensions) of definitions to install. In the `sofin install base` example above, the "base" part is just a name of a list that includes mentioned software definitions. Sofin will automatically recognize which one is given. Most common usage of Sofin is: `sofin install softwarename` and `sofin remove softwarename`. Most of the time, it's the only thing you want to do. "You want to type one command and get your software" – this was one of my major thoughts when I was starting to write Sofin. But there's one more important thing to mention. By default, software lists and definitions (with software patches) are just a plain bz2 archive put on an http server. Sofin isn't updating those definitions on each run. To get fresh definitions from a server, you need to run `sofin update`. The thing is-- they're only user side definitions. Each user may have different local versions of them, and these definitions do not affect other users. One more common feature of Sofin, is a partial software upgrade. For example: to upgrade libffi in a Ruby bundle, you need to call `sofin upgrade libffi ruby`. It might be confusing, but the command works as it would in natural language: "use sofin to upgrade libffi dependency in Ruby bundle". Sofin will automatically detect rebuilt dependency and invoke a rebuild of ruby itself. It's important to mention that the upgrade process isn't perfect yet – it's one of the features that might fail with certain definitions, so please keep that in mind. It tries to look for patterns of updated definition names and removes matching files from a bundle before doing the upgrade, but it's just very tough to test all the possibilities. In case of a failure, just rebuild the whole software from scratch.

Implemented utility commands and hidden features

Sofin comes as a shell command. The launcher itself is written in C++ to fully support lock file functionality which is used by Sofin under the hood to avoid some issues (mostly to solve launching two conflicting commands at once). If one software build is already in progress, the second instance of Sofin will wait until the first is finished.

Here's a list of additional Sofin commands with short explanations:

- `sofin list` – Lists all installed bundles owned by the current user. Use `sofin fulllist` to get a list with dependencies included.
- `sofin available` – Shows the list of all definitions available to install.
- `sofin vars` – Generates the `sh`-compliant dump of ENV values, based on the currently installed software.
- `sofin log` – Probably the most useful command if you want to see what's going on under Sofin's hood. It will show the software installation progress log, including all commands invoked during the process.
- `sofin ver` – Shows Sofin's version.
- `sofin outdated` – Shows software bundles installed by Sofin that are outdated.
- `sofin clean` – Removes source packages cache. It also clears the installation log.
- `sofin dependencies` – One of the rarely used features that might be used for dedicated software. It reads `$(cwd)/.dependencies` file as software list, and installs software from it.
- `DEBUG=true sofin anycommand` – Turns on debug logger if you're really interested about details (prints in magenta to the standard output).

Some comments about the development process and Sofin's issues

Sofin isn't the ideal software, but it has already proven to be very useful in several production environments. Here's the thing: I don't want to hide any of its pitfalls because I don't have to.

Sofin is built to always be production-ready software. This means that, if your software doesn't build, then in the vast majority of cases, it will be caused either by an error in definition or by an issue with the software itself.

Sofin's core has been almost untouched since the beginning of the project and is only extended with new features from time to time (usually on request).

But of course, there are a few limitations. First of all, the default definition source is placed on my private HTTP server. If you want to host your own definitions, you'd need to reconfigure Sofin for your needs. The second major pitfall is that it requires XQuartz on OS X hosts to build some definitions. I couldn't find a better solution for Mac OS X without Darwin-specific hacks on X11.

On the Web

- Sofin project page
- TheSS project page (currently in development)
- My website

Glossary

- KISS – the acronym of Keep It Simple Stupid. Methodology to software development with simplicity in mind.
- FHS – Filesystem Hierarchy Standard.
- POSIX – compatibility standard between operating systems (used by BSD, OS X and Linux).

TheSS, Sofin-based software management tool

Parallel with Sofin, the second project called TheSS, is under heavy development. It was closed source, but this year I made it public (BSD-licensed as well).

In short, it uses software built by Sofin to easily perform software deployment (including the support for web applications of any kind).

Summary

Sofin is free software. The content of this article covers only a part of the design patterns used internally. Feel free to support this project in any way. I'm easy to find – there's only one dmilith.

I'm open to suggestions and improvement ideas. Please feel free to contribute. If you want to get more details, just find me online, and I'll try to explain every detail you might want to know about Sofin and TheSS.

DANIEL (DMILITH) DETTLAFF

Sysadmin of several medium companies in Poland. Enterprise and cloud hater. Currently working at Monterail.com, LLC. Working on building self-healing, self-manageable, distributed, AI-driven systems. He makes new ideas real while systems are automatically doing his work. Constantly learning about how to become a good software architect. Musician, lyricist, and writer in his free time. (Thanks to Dominik Porada and Michał Hewelt for help with my horrible English grammar)

What do all these have in common?

They all use Nipper Studio

to audit their firewalls, switches & routers

Nipper Studio is an award winning configuration auditing tool which analyses vulnerabilities and security weaknesses. You can use our point and click interface or automate using scripts. Reports show:

1) Severity of the Threat & Ease of Resolution

2) Configuration Change Tracking & Analysis

3) Potential Solutions including Command Line Fixes to resolve the Issue

Nipper Studio doesn't produce any network traffic, doesn't need to interact directly with devices and can be used in secure environments.

SME pricing from £650 scaling to enterprise level

evaluate for free at www.titania.com

2012 Computing Security Awards WINNER Enterprise Security Solution of the Year

2012 Computing Security Awards WINNER Network Security Solution of the Year

2012 Computing Security Awards Runner-up SME Security Solution of the Year

titania nipper® STUDIO

www.titania.com
T: +44 (0) 1905 888785

A Backup Server

with FreeBSD for Mixed Networks in SOHO Environment

Backing up servers and clients is an essential task that should be carried out on a regular basis as it helps prevent the loss of data. The backup tasks can be performed in two different ways, using automated software or by running the software manually. It is essential, however, to make sure that the backups are working properly and running on a regular basis.

What you will learn…

- Using FUSE to mount remote filesystems
- Using ssh keys to remote access ssh server without passwords
- Back up mixed networks with FreeBSD

What you should know…

- Minimum scripting knowledge
- Minimum networking concepts knowledge

The server backups may include data from external users. For a company such as a web hosting service, it is essential to have a copy of the sites and databases of the customers as they are completely reliant on the web host to keep their sensitive information safe. Backups are fundamentally necessary for disaster recovery. In the case of a server failure, a backup makes it possible to retrieve data from the server offsite.

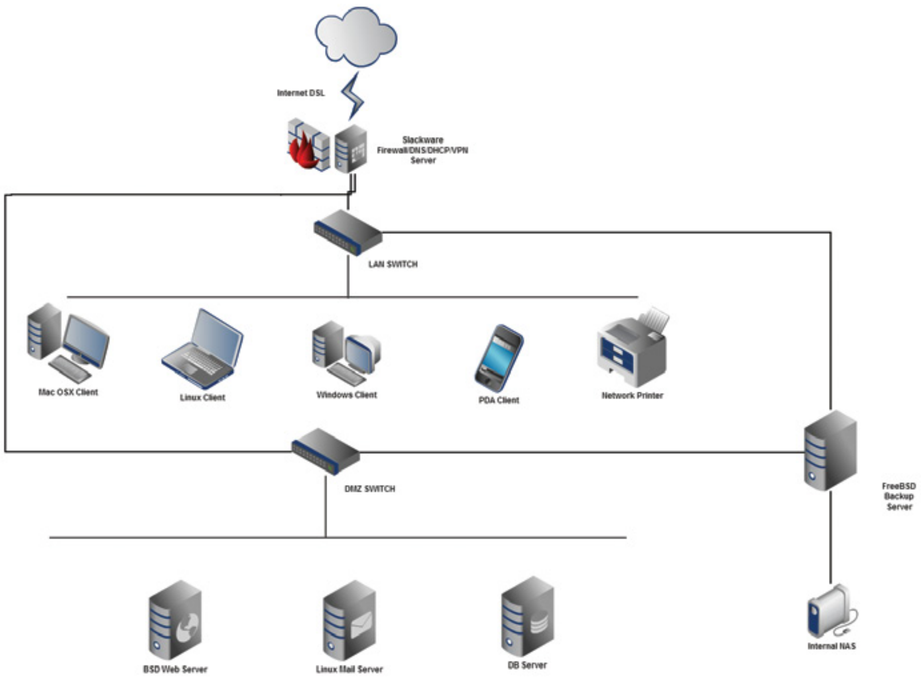


Figure 1. *Our Test LAN and DMZ scenario*

Having backups always available makes it easier to migrate data if you intend to move the data from an old server to a new one that is in the DMZ itself, or at another location.

From the examples above, it is easy to understand how important it is to make backups of servers and clients. You could even say that this is a fundamental task for business (Figure 1).

FreeBSD once again demonstrates its flexibility. As you will see later in the article, it will be possible to create a dedicated server that will allow you to create backups of Windows machines via the CIFS protocol, Unix machines via SSHFS, and either OS using an FTP server with curlftpfs.

It is also possible to make backup copies of databases, using scripts integrated into the task (in particular, you will see an example using mysqldump).

Everything will be handled in an automated manner using cron jobs managed by the software rsnapshot, the heart of the backup system.

Listing 1. *Packages Setup*

```
pkg_add -r -v samba36-smbclient
pkg_add -r -v mysql51-client
pkg_add -r- v postgresql91-client
pkg_add -r -v fuse
pkg_add -r -v fusefs-kmod
pkg_add -r -v fusefs_sshfs
pkg_add -r -v fusefs-curlftpfs
pkg_add -r -v rsnapshot
pkg_add -r -v ssh-copy-id
```

Listing 2. *Packages Setup*

```
portsnap fetch
portsnap update
cd /usr/ports/ftp/vsftpd
make install clean
```

Listing 3. *Rc.conf setup*

```
fbsd-bkpsrv#cat /etc/rc.conf

#hostname="fbsd-bkpsrv.localdomain"
hostname="fbsd-bkpsrv"
ifconfig_em0="DHCP"
ifconfig_em1="DHCP"
keymap="it.iso"
moused_enable="YES"
sshd_enable="YES"
fusefs_enable="YES"
vsftpd_enable="YES"
```

To create a good backup strategy, one needs to do an analysis of the resources that store the data to be protected. In a SOHO network, this can be a LAN and a DMZ that exports outside services, such as a web server. Imagining that your clients will use Windows or Mac OSX as well as Linux and FreeBSD, CIFS is a good way to share data in a simple manner within the network itself.

If you have servers located in the DMZ such as SSH, FTP and SQL, you can mount these resources as local folders within the backup server and then manage the backup pool with the appropriate cron job.

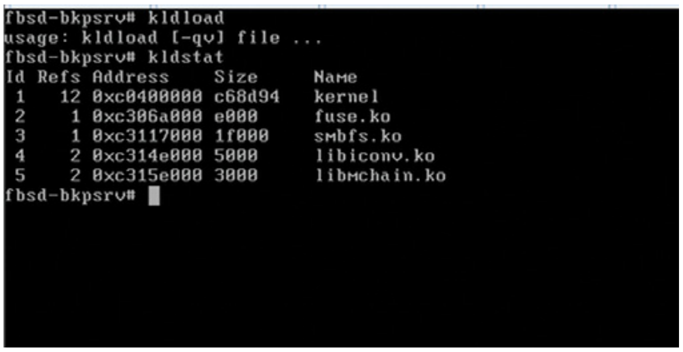
Backup Server Packages Installation

It was decided to use the 8.3 version of FreeBSD, due to some unresolved bugs on the port of curlftpfs. In addition, for easy restoration of the data, the daemon Vsftpd has to be activated on an ftp server. Of course, this is only one of the possible solutions. To manage the backup of a mixed network, one needs to install a set of packages on our server: Listing 1.

In order to have the latest version of vsftpd, it needs to be installed via ports (Listing 2).

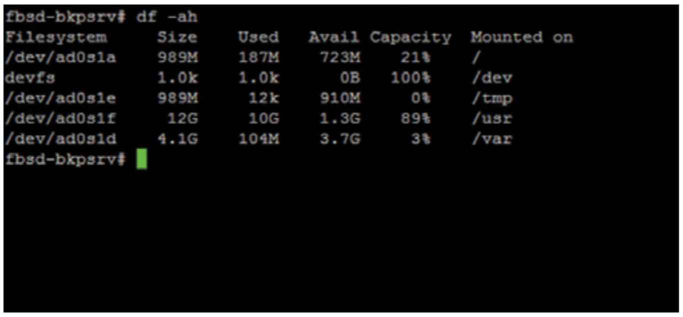
Once you have installed all the packages, you're going to do all the individual configurations. In particular, it will be necessary to set the rc.conf file as follows, in relation to the networks of Figure 1 and Listing 3.

Once the server has restarted, you will see the new active modules in the kernel as shown in Figure 2.



```
fbsd-bkpsrv# kldload
usage: kldload [-qv] file ...
fbsd-bkpsrv# kldstat
Id Refs Address    Size     Name
 1   12 0xc0400000 c68d94   kernel
 2    1 0xc306a000 e000     fuse.ko
 3    1 0xc3117000 1f000    smbfs.ko
 4    2 0xc314e000 5000     libiconv.ko
 5    2 0xc315e000 3000     libmchain.ko
fbsd-bkpsrv#
```

Figure 2. *Kernel Modules loaded after reboot*



```
fbsd-bkpsrv# df -ah
Filesystem     Size    Used   Avail Capacity  Mounted on
/dev/ad0s1a    989M    187M    723M    21%    /
devfs          1.0k    1.0k      0B   100%    /dev
/dev/ad0s1e    989M     12k    910M     0%    /tmp
/dev/ad0s1f     12G     10G    1.3G    89%    /usr
/dev/ad0s1d    4.1G    104M    3.7G     3%    /var
fbsd-bkpsrv#
```

Figure 3. *Our backup server original mount points*

One of the active modules will be "fuse.ko", which has a particular function in that it allows non-privileged users of a system to create their own file system without the need to write code in kernel level. This is particularly useful for writing a virtual file system, which does not actually store the data on its own, but mediates between the user and the underlying real filesystem. FUSE is the module that allows you to use SSHFS filesystem with CIFS and curlftpfs (Figure 3).

SSH login without a password for SSHFS on FreeBSD

In order to create tasks for automatic backup via SSH, it is necessary to establish a certain level of trust between the computers. To do this, one can resort to the use of pairs of keys. First create a key pair on the local machine for user `root`: Listing 4.

We call the machine `LOCAL_MACHINE` and your username will be `root`. Now you need to copy the public key on the remote machine (`remote_machine`) with 'root': Listing 5.

From now on, the user `root@remote_machine` will trust user `root@LOCAL_MACHINE` and it will allow access without asking for a password. If that fails, you will have to edit `/etc/ssh/sshd_config` on the remote machine, add the following lines, and restart ssh: Listing 6.

We have already installed packages: Listing 7, and enabled FUSE for system start up via rc.conf (Listing 8).

After the reboot, if one can ssh to the `remote_machine`, one can at least mount their own home directory via SSH (Listing 9).

FreeBSD mounting remote CIFS resources

We have just loaded the fuse module and installed the smbclient package which provides the utility `mount_smbfs` that will mount a share from a remote server using SMB/CIFS protocol. You can easily mount a NAS share using the following syntax: Listing 10. Where:

- `NETBIOSNAME`: connection to the FQDN or IP of the remote workstation or server
- `USERNAME`: the login user name.
- `NETBIOSNAME –` : NETBIOS Server Name.
- `/data –` : CIFS share name.
- `/mnt/net/NETBIOSNAME –` : local mount point directory.

Listing 4. *Ssh daemon and client key configuration*

```
root@local_machine:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/root/.ssh/
                    id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/root/.ssh/
                    id_rsa.
Your public key has been saved in /home/root/.ssh/id_
                    rsa.pub.
The key fingerprint is:
3d:23:ab:7c:f3:27:46:42:6b:76:e2:41:5e:79:b8:d1 root@
                    local_machine
The key's randomart image is:
+--[ RSA 2048]----+
|          +      |
|        o = Z    |
|       + o +     |
|        O.o      |
|       =K++      |
|        oo o     |
|       ..        |
|      o.o .      |
|     .o=.o       |
+-----------------+
```

Listing 5. *Ssh daemon and client key configuration*

```
ssh-copy-id -i ~/.ssh/id_rsa.pub root@remote_machine
```

Listing 6. *Ssh daemon and client key configuration*

```
RSAAuthentication yes
PubkeyAuthentication yes
```

Listing 7. *Ssh daemon and client key configuration*

```
fusefs-kmod
fusefs-sshfs
```

Listing 8. *Ssh daemon and client key configuration*

```
# enable File System in User Space
fusefs_enable="YES"
```

Listing 9. *Ssh daemon and client key configuration*

```
sshfs username@remote_machine ~/mount_point
```

Listing 10. *Samba client configuration*

```
# mount_smbfs -I NETBIOSNAME //USERNAME@NETBIOSNAME/data
                    /mnt/net/NETBIOSNAME
```

Listing 11. *Samba client configuration*

```
[NETBIOS_NAME:USERNAME]
password=PASSWORD
```

In this way, however, it will be prompted for the password at each login. To avoid the password prompt, we need to create a `~/.nsmbrc` file as follows:

```
# emacs ~/.nsmbrc
```

Enter the username and password as follows: Listing 11.

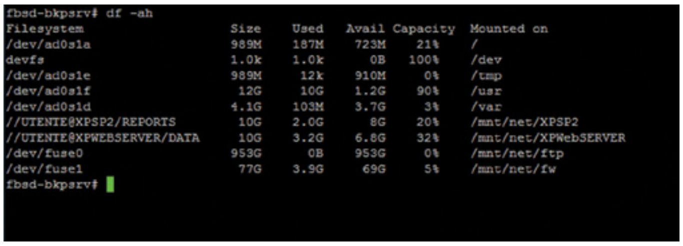
Mount the remote CIFS folder as follows: Listing 12.

With the `-N` option, one forces the system to read the `~/.nsmbrc` file for additional configuration parameters and a password.

How to use FTP filesystem on FreeBSD with CurlFtpFS

It often happens that some web hosting companies do not offer shell access (SSH or Telnet) to your shared hosting account for security reasons. This makes it more difficult to do regular maintenance of the file system on your web server. Despite that the use of a normal ftp client is sufficient for the majority of cases, some people still prefer to manipulate files directly using standard Unix. This is possible thanks to curlftpfs which allows you to mount a remote FTP as a standard file system on the Unix operating system, and this allows one to do backup tasks. If curlftpfs is installed, you only need to do these steps to locally mount a remote folder: Listing 13.

`user:pass` – is the username and password to log into the ftp account. After that, you can change your working directory to the mount-point and use the regular Unix utilities to work on the files that are normally accessible with the FTP protocol. To unmount it, one can use the command: Listing 14.



```
fbsd-bkpsrv# df -ah
Filesystem                      Size    Used   Avail Capacity  Mounted on
/dev/ad0s1a                     989M    187M    723M    21%    /
devfs                           1.0k    1.0k      0B   100%    /dev
/dev/ad0s1e                     989M     12k    910M     0%    /tmp
/dev/ad0s1f                      12G     10G    1.2G    90%    /usr
/dev/ad0s1d                     4.1G    103M    3.7G     3%    /var
//UTENTE@XPSP2/REPORTS           10G    2.0G      8G    20%    /mnt/net/XPSP2
//UTENTE@XPWEBSERVER/DATA        10G    3.2G    6.8G    32%    /mnt/net/XPWebSERVER
/dev/fuse0                      953G      0B    953G     0%    /mnt/net/ftp
/dev/fuse1                       77G    3.9G     69G     5%    /mnt/net/fw
fbsd-bkpsrv#
```

Figure 4. *Our server with remote mount points active*

In this way, however, it is possible to read clear text passwords in log files. To avoid this, we need to create a `~/.netrc` file as follows: Listing 15 and enter host, username and password as follows: Listing 16, Figure 4 and Figure 5.

A FreeBSD local FTP Server to restore backup data

Two words on FTP:

File Transfer Protocol (FTP) is a TCP protocol for exchanging files between computers. It does not use encryption for user credentials and, unless merged into an SSL connection, the data is transmitted in the clear and can be easily intercepted. FTP works on a client/server model and the server component is called an FTP daemon. It is always listening for FTP requests from remote clients. If you have a request, it handles the authentication, keeps the connection alive for the duration of the session, and executes the commands sent by the FTP client. Access to an FTP server can be managed either in an anonymous or an authenticated mode. In the Anony-

Listing 12. *Samba client configuration*

```
# mount_smbfs -N -I NETBIOSNAME //USERNAME@
   NETBIOSNAME/data /mnt/net/NETBIOSNAME
```

Listing 13. *Curlftpfs client configuration*

```
mkdir /mnt/net/ftp
curlftpfs -o allow_other ftp://user:pass@ftp_host_name
```

Listing 14. *Curlftpfs client configuration*

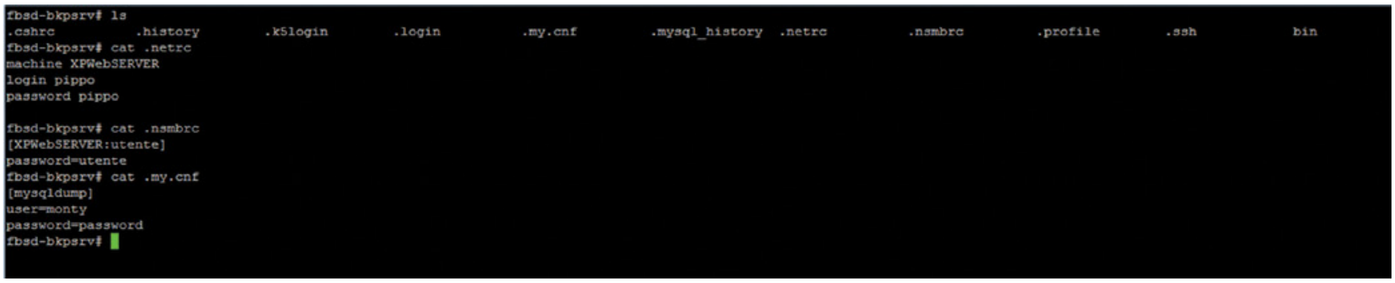
```
umount mountpoint
```

Listing 15. *Curlftpfs client configuration*

```
# emacs ~/.netrc
```

Listing 16. *Curlftpfs client configuration*

```
machine ftp_host_name
login user
password pass
```



```
fbsd-bkpsrv# ls
.cshrc          .history        .k5login        .login          .my.cnf         .mysql_history  .netrc          .nsmbrc         .profile        .ssh            bin
fbsd-bkpsrv# cat .netrc
machine XPWebSERVER
login pippo
password pippo

fbsd-bkpsrv# cat .nsmbrc
[XPWebSERVER:utente]
password=utente
fbsd-bkpsrv# cat .my.cnf
[mysqldump]
user=monty
password=password
fbsd-bkpsrv#
```

Figure 5. *All credential files*

mous mode, remote clients can access the FTP server using the default user account called "anonymous" or "ftp" and by sending an e-mail address as the password. In the authenticated mode, a user must have an account and a password. User access to the FTP server directories and files depends on the permissions defined for the account used to login. As a general rule, the FTP daemon will hide the root directory of the FTP server and change the FTP home directory. This hides the rest of the file system from the remote sessions.

vsftpd – FTP Server Configuration

Setting up an FTP server is beyond the scope of this article; however you should choose to create a basic service that enables access protected by username and password in the local networks (Listing 17).

Listing 17. *Vsftpd server configuration*

```
# cat /usr/local/etc/vsftpd.conf
listen=YES
anonymous_enable=NO
anon_upload_enable=NO
anon_mkdir_write_enable=NO
background=YES
local_enable=YES
write_enable=NO
xferlog_enable=YES
ftpd_banner=Welcome to LAN FTP service
chroot_list_enable=YES
chroot_list_file=/usr/local/etc/vsftpd.chroot_list
userlist_enable=YES
userlist_deny=NO
allow_writeable_chroot=YES
```

Listing 18. *Vsftpd server configuration*

```
# cat /usr/local/etc/vsftpd.chroot_list
utente
backup
rsnapshot
```

Listing 19. *Vsftpd server configuration*

```
# cat /usr/local/etc/vsftpd.user_list
utente
backup
rsnapshot
```

Listing 20. *Vsftpd server configuration*

```
# cat /usr/local/etc/vsftpd.ftpusers
```

So, as configured, vsftpd accepts connections from both the LAN and DMZ. It also allows access to users listed in the file `vsftpd.user_list` and access to data to those contained in the file `vsftpd.chroot_list` (Listing 18-20).

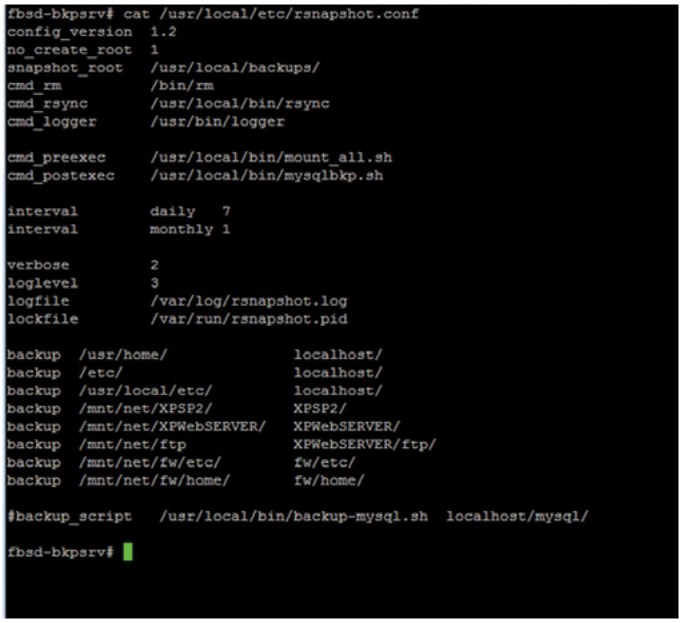
The file `/usr/local/etc/vsftpd.ftpusers` is empty by default. Access to the FTP server is read-only, because it is not good to be able to accidentally erase our backup file!

Rsnapshot with FreeBSD to manage backup data pools

For anyone who has never heard of rsnapshot, it is a program that allows you to create "snapshots" of the filesystem. You can take incremental snapshots of local and remote file systems for any number of machines. Snapshots of local file systems are handled with rsync, a milestone in Unix backup tools. Secure remote connections are treated with rsync over ssh while anonymous rsync connections simply use an rsync server. Both remote and local transfers depend on rsync. Rsnapshot saves much more disk space than you might imagine. The amount of space required is about the size of a full backup, plus one copy of each additional file that is changed. Rsnapshot makes extensive use of hard links, so if the file does not change, the next snapshot is simply a hard link to the exact same file.

The architecture of the server backup made allows you to "see" the remote folders as local, and for rsnapshot and rsync, it is easy to manage pool copies of this kind.

Starting from a root directory, rsnapshot allows you to create a number of subfolders by date. Each of these sub-



```
fbsd-bkpsrv# cat /usr/local/etc/rsnapshot.conf
config_version  1.2
no_create_root  1
snapshot_root   /usr/local/backups/
cmd_rm          /bin/rm
cmd_rsync       /usr/local/bin/rsync
cmd_logger      /usr/bin/logger

cmd_preexec     /usr/local/bin/mount_all.sh
cmd_postexec    /usr/local/bin/mysqlbkp.sh

interval        daily   7
interval        monthly 1

verbose         2
loglevel        3
logfile         /var/log/rsnapshot.log
lockfile        /var/run/rsnapshot.pid

backup  /usr/home/              localhost/
backup  /etc/                   localhost/
backup  /usr/local/etc/         localhost/
backup  /mnt/net/XPSP2/         XPSP2/
backup  /mnt/net/XPWebSERVER/   XPWebSERVER/
backup  /mnt/net/ftp            XPWebSERVER/ftp/
backup  /mnt/net/fw/etc/        fw/etc/
backup  /mnt/net/fw/home/       fw/home/

#backup_script   /usr/local/bin/backup-mysql.sh  localhost/mysql/

fbsd-bkpsrv#
```

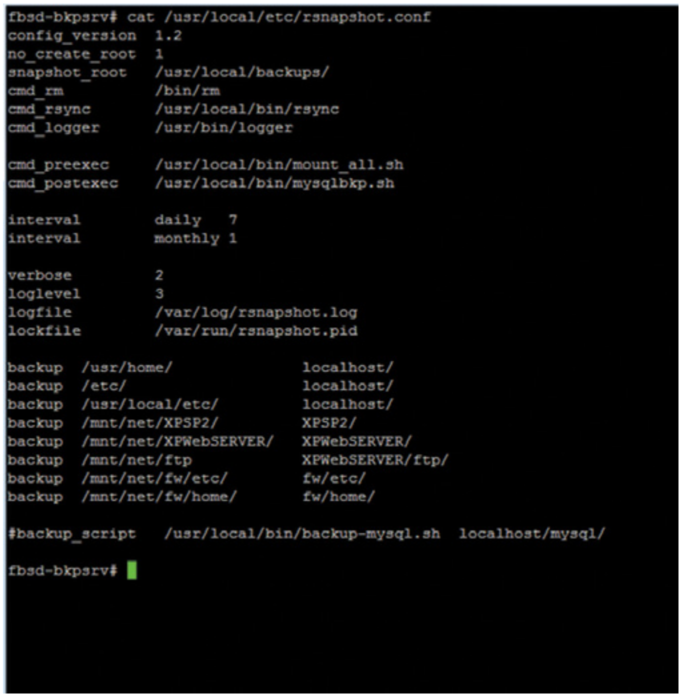
Figure 6. *Our server rsnapshot.conf*

folders, organized by host, will contain the data for incremental backups, as shown in Figure 5.

The configuration file presented performs one cron job daily and one monthly, so you have a valid backup set. It is crucial to remember that the configuration file only allows tabs as a separator character (Listing 21 and Figure 6).

Within the configuration, the time intervals for performing backups are set using the parameter "interval". They do nothing but run the cron job like the one below: Listing 22.

Within the configuration, you can run custom scripts. (An example would be to mount remote file systems, another would be to make backups of a MySQL server.) The important thing is, that these files are on the inside of the paths in the PATH variable of the system (eg. `/usr/local/bin`).

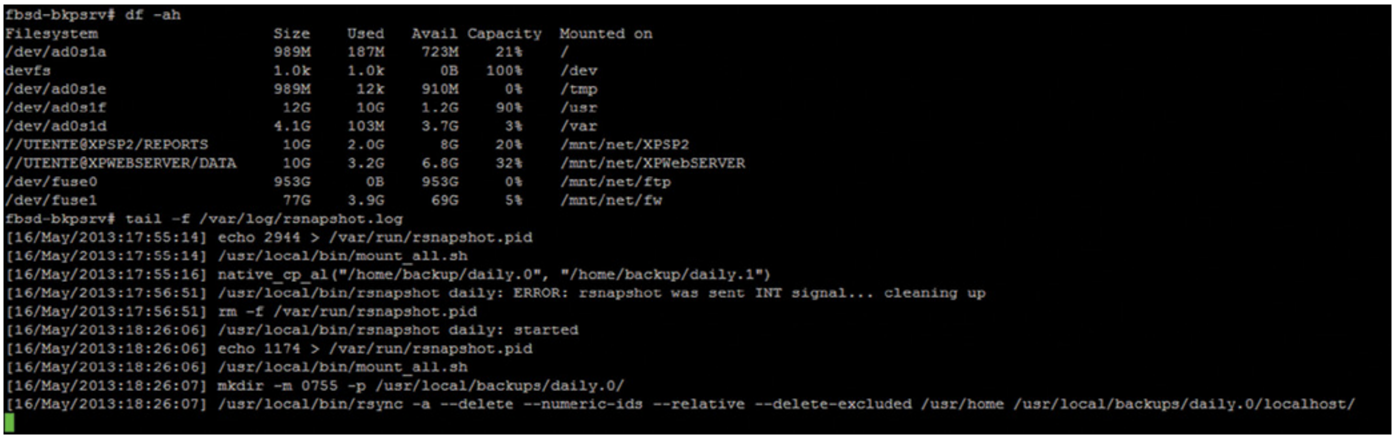


```
fbsd-bkpsrv# cat /usr/local/etc/rsnapshot.conf
config_version  1.2
no_create_root  1
snapshot_root   /usr/local/backups/
cmd_rm          /bin/rm
cmd_rsync       /usr/local/bin/rsync
cmd_logger      /usr/bin/logger

cmd_preexec     /usr/local/bin/mount_all.sh
cmd_postexec    /usr/local/bin/mysqlbkp.sh

interval        daily   7
interval        monthly 1

verbose         2
loglevel        3
logfile         /var/log/rsnapshot.log
lockfile        /var/run/rsnapshot.pid

backup  /usr/home/              localhost/
backup  /etc/                   localhost/
backup  /usr/local/etc/         localhost/
backup  /mnt/net/XPSP2/         XPSP2/
backup  /mnt/net/XPWebSERVER/   XPWebSERVER/
backup  /mnt/net/ftp            XPWebSERVER/ftp/
backup  /mnt/net/fw/etc/        fw/etc/
backup  /mnt/net/fw/home/       fw/home/

#backup_script   /usr/local/bin/backup-mysql.sh  localhost/mysql/

fbsd-bkpsrv#
```

Figure 7. *Our server when a backup pool is running (logfile)*

Listing 21. *rsnapshot configuration*

```
# cat /usr/local/etc/rsnapshot.conf
config_version  1.2
no_create_root  1
snapshot_root   /usr/local/backups/
cmd_rm          /bin/rm
cmd_rsync       /usr/local/bin/rsync
cmd_logger      /usr/bin/logger

cmd_preexec     /usr/local/bin/mount_all.sh
cmd_postexec    /usr/local/bin/mysqlbkp.sh

interval        daily   7
interval        monthly 1

verbose         2
loglevel        3
logfile        /var/log/rsnapshot.log
lockfile       /var/run/rsnapshot.pid

backup  /usr/home/              localhost/
backup  /etc/                   localhost/
backup  /usr/local/etc/         localhost/
backup  /mnt/net/XPSP2/         XPSP2/
backup  /mnt/net/XPWebSERVER/   XPWebSERVER/
backup  /mnt/net/ftp            XPWebSERVER/ftp/
backup  /mnt/net/fw/etc/        fw/etc/
backup  /mnt/net/fw/home/       fw/home/

#backup_script   /usr/local/bin/backup-mysql.sh
                    localhost/mysql/
```



```
fbsd-bkpsrv# df -ah
Filesystem                    Size    Used   Avail Capacity  Mounted on
/dev/ad0s1a                   989M    187M    723M    21%    /
devfs                         1.0k    1.0k      0B   100%    /dev
/dev/ad0s1e                   989M     12k    910M     0%    /tmp
/dev/ad0s1f                    12G     10G    1.2G    90%    /usr
/dev/ad0s1d                   4.1G    103M    3.7G     3%    /var
//UTENTE@XPSP2/REPORTS         10G    2.0G      8G    20%    /mnt/net/XPSP2
//UTENTE@XPWEBSERVER/DATA      10G    3.2G    6.8G    32%    /mnt/net/XPWebSERVER
/dev/fuse0                    953G      0B    953G     0%    /mnt/net/ftp
/dev/fuse1                     77G    3.9G     69G     5%    /mnt/net/fw
fbsd-bkpsrv# tail -f /var/log/rsnapshot.log
[16/May/2013:17:55:14] echo 2944 > /var/run/rsnapshot.pid
[16/May/2013:17:55:14] /usr/local/bin/mount_all.sh
[16/May/2013:17:55:16] native_cp_al("/home/backup/daily.0", "/home/backup/daily.1")
[16/May/2013:17:56:51] /usr/local/bin/rsnapshot daily: ERROR: rsnapshot was sent INT signal... cleaning up
[16/May/2013:17:56:51] rm -f /var/run/rsnapshot.pid
[16/May/2013:18:26:06] /usr/local/bin/rsnapshot daily: started
[16/May/2013:18:26:06] echo 1174 > /var/run/rsnapshot.pid
[16/May/2013:18:26:06] /usr/local/bin/mount_all.sh
[16/May/2013:18:26:07] mkdir -m 0755 -p /usr/local/backups/daily.0/
[16/May/2013:18:26:07] /usr/local/bin/rsync -a --delete --numeric-ids --relative --delete-excluded /usr/home /usr/local/backups/daily.0/localhost/
```

Figure 8. *Our server when a backup pool is completed*

To verify that the setup is correct, you can run rsnapshot with the "configtest" options, and if all is well, it should say – `Syntax OK`: Listing 23.

With this configuration, you backup local folders:

```
/usr/home/              localhost/
/etc/                   localhost/
/usr/local/etc/         localhost/
```

And remote folders are locally mounted:

```
/mnt/net/XPSP2/         XPSP2/
/mnt/net/XPWebSERVER/   XPWebSERVER/
/mnt/net/ftp            XPWebSERVER/ftp/
/mnt/net/fw/etc/        fw/etc/
/mnt/net/fw/home/       fw/home/
```

Seven times a week and once a month (Figure 7 and Figure 8).

To automate all, you can create a small script "mount_all" of which this is a simple example: Listing 24.

Listing 22. *rsnapshot configuration*

```
# cat /etc/periodic/daily/001.backup
/usr/local/bin/rsnapshot daily > /tmp/rsnapshot.out 2>&1
                    || cat \ /tmp/rsnapshot.out | mail -s
                    "daily backups failed on `hostname`"
                    \ antofrage@xxx.xx
```

Listing 23. *rsnapshot configuration*

```
# rsnapshot configtest
Syntax OK
```

Listing 24. *scripts configuration*

```
#cat /usr/local/bin/mount_all.sh

#!/bin/sh
if [ "$(ls -A /mnt/net/XPSP2)" ]; then
     umount /mnt/net/XPSP2
     mount_smbfs -N -I XPSP2 //utente@XPSP2/reports /
                    mnt/net/XPSP2/
else
    echo "/mnt/net/XPSP2 is Empty"
     mount_smbfs -N -I XPSP2 //utente@XPSP2/reports /
                    mnt/net/XPSP2/
fi

if [ "$(ls -A /mnt/net/XPWebSERVER)" ]; then
     umount /mnt/net/XPWebSERVER
     mount_smbfs -N -I XPWebSERVER //utente@XPWebSERVER/
                    data /mnt/net/XPWebSERVER/
else
    echo "/mnt/net/XPWebSERVER is Empty"
     mount_smbfs -N -I XPWebSERVER //utente@XPWebSERVER/
                    data /mnt/net/XPWebSERVER/
fi

if [ "$(ls -A /mnt/net/ftp)" ]; then
     umount /mnt/net/ftp
     curlftpfs -o allow_other XPWebSERVER /mnt/net/ftp
else
    echo "/mnt/net/ftp is Empty"
    curlftpfs -o allow_other XPWebSERVER /mnt/net/ftp
fi

if [ "$(ls -A /mnt/net/fw)" ]; then
     umount /mnt/net/fw
     sshfs root@fw:/ /mnt/net/fw/
else
    echo "/mnt/net/fw is Empty"
    sshfs root@fw:/ /mnt/net/fw/
fi
```

Listing 25. *scripts configuration*

```
#cat /usr/local/bin/mysqlbkp.sh
#!/bin/sh
USER0="monty"
HOST0="XPWebSERVER"
mysqldump -u$USER0 -h $HOST0 --all-databases | bzip2 -c
                   > \
mysql_`date +%Y-%m-%d`_bkp.sql.bz2
```

Listing 26. *scripts configuration*

```
# cat /etc/hosts

# $FreeBSD$
# Host Database

::1                     localhost localhost.my.domain
                   myname.my.domain
127.0.0.1               localhost localhost.my.domain
                   myname.my.domain

# HOSTS
192.168.254.3            XPWebSERVER
192.168.0.3              XPSP2
```

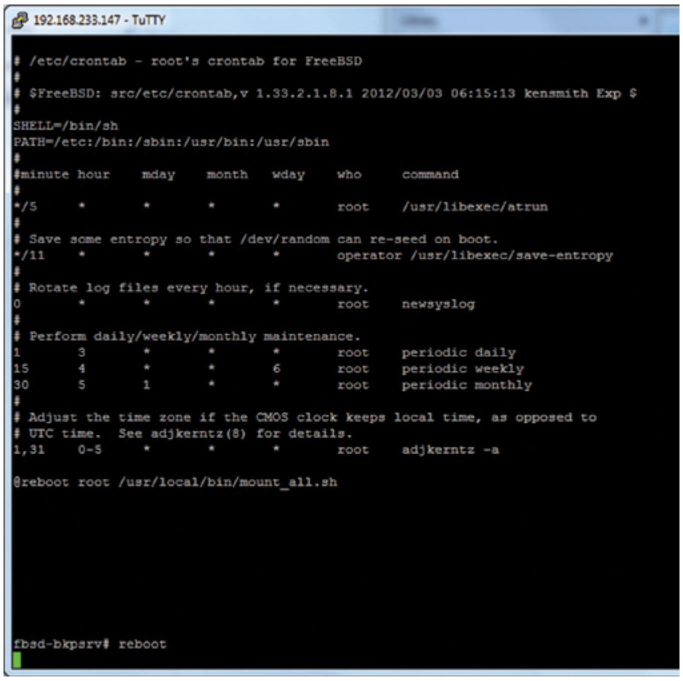
For completeness, here is a simple backup script to a MySQL server accessible from the LAN: Listing 25 and Figure 9.

A little foresight

To simplify the work, it can be useful to map the names of the servers and clients in the hosts file of the backup server. This is so that in case of a malfunction of the DNS server, everything will still work (Listing 26).

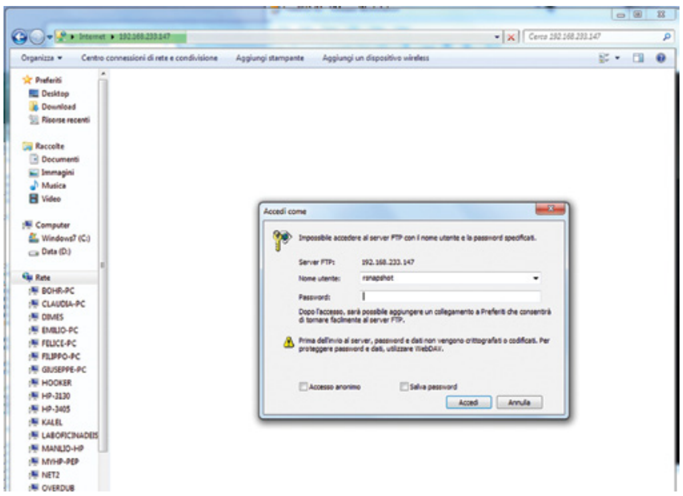
How to restore backup data?

All that you need is an FTP client (FileZilla, FTP CLI, Explorer, Firefox, etc.)! As shown in Figure 10, simply con-



```
# /etc/crontab - root's crontab for FreeBSD
#
# $FreeBSD: src/etc/crontab,v 1.33.2.1.8.1 2012/03/03 06:15:13 kensmith Exp $
#
SHELL=/bin/sh
PATH=/etc:/bin:/sbin:/usr/bin:/usr/sbin
#
#minute hour    mday    month   wday    who     command
#
*/5     *       *       *       *       root    /usr/libexec/atrun
#
# Save some entropy so that /dev/random can re-seed on boot.
*/11    *       *       *       *       operator /usr/libexec/save-entropy
#
# Rotate log files every hour, if necessary.
0       *       *       *       *       root    newsyslog
#
# Perform daily/weekly/monthly maintenance.
1       3       *       *       *       root    periodic daily
15      4       *       *       6       root    periodic weekly
30      5       1       *       *       root    periodic monthly
#
# Adjust the time zone if the CMOS clock keeps local time, as opposed to
# UTC time.  See adjkerntz(8) for details.
1,31    0-5     *       *       *       root    adjkerntz -a

@reboot root /usr/local/bin/mount_all.sh




fbsd-bkpsrv# reboot
```

Figure 9. *Our server when a main crontab*



Accedi come

Impossibile accedere al server FTP con il nome utente e la password specificati.

Server FTP: 192.168.233.147

Nome utente: rsnapshot

Password:

Dopo l'accesso, sarà possibile aggiungere un collegamento a Preferiti che consentirà di tornare facilmente al server FTP.

Prima dell'invio al server, password e dati non vengono crittografati o codificati. Per proteggere password e dati, utilizzare WebDAV.

Figure 10. *Logging into Ftp Service of Backup Server*

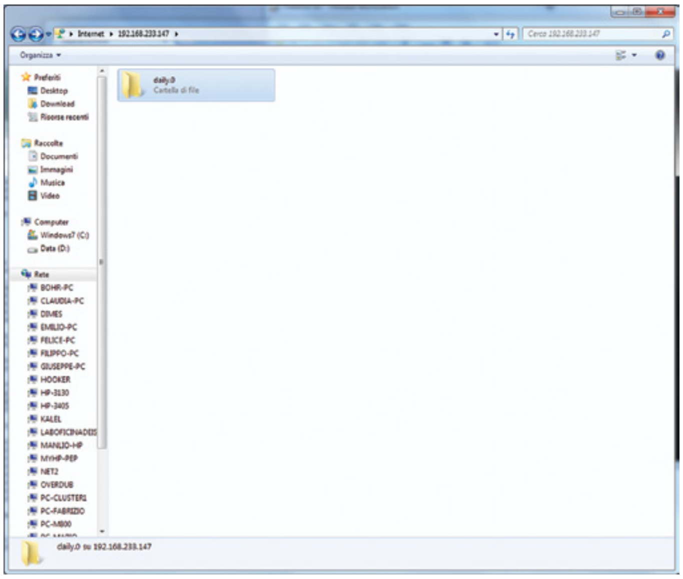


Figure 11. *Browsing folders*

nect to the local FTP server and check the date of the file or folder you want to restore.

A simple copy/paste et voila (Figure 11). :)

With this article, we wanted to create an efficient and robust backup server designed for SOHO. Enterprise solutions in need of more advanced features can use FreeNAS, which is also based on BSD.

ANTONIO FRANCESCO GENTILE

lives in Italy, Calabria. He is a software and network engineer. He works with National Center of Research (ICAR) area networking in Cosenza as network manager, with the "Culture Lab" http://culture.deis.unical.it Department of Telematics at University of Calabria, and with the computer science associations "Hacklab Cosenza" http://hacklab.cosenzainrete.it/ and "Verde Binario" http://www.verdebinario.org/ and is a freelance columnist for Italian magazines "Linux&C" http://www.oltrelinux.com/ and "Linux Magazine" http://www.linux-magazine.it/.

FreeBSD on Rails

Ruby on Rails is a powerful Web framework. It makes application prototyping a breeze, in a few days. Installing it is quite trivial if you know the pitfalls.

What you will learn…
- Installing all Rails-related development tools,
- Setting up a web application's scaffolding,
- Debugging your application.

What you should know…
- Configuring your HTTP server (Nginx, Apache…),
- Setting up your database management system,
- HTML.

I also strongly advocate that you get to know/learn Git. Nearly all Ruby on Rails-related tools, as well as their documentation, are hosted on GitHub.

Installing

Many tools may help you optimize your Ruby on Rails workflow:

- *rbenv* helps managing several ruby versions on a single computer,
- *gems* are basically ruby programs or libraries,
- *bundler* manages project-specific gems,
- *rake* executes scripts which are part of our application.

All these tools are included within *rbenv,* which is why I always use this tool no matter how many projects I am working on (which means even just one).

You will be using those all the time, regardless of your environment, be it personal or professional. Getting used to the tools everybody uses is always a good approach.

Side note, my shell of choice is *zsh*. Adapt the listings below to suit your needs – for instance, if you use *bash*, replace `~/.zshrc` with `~/.bashrc` in the listings.

Dependencies

To get ourselves started, please first install (with *root/sudo*) these dependencies:

```
# pkg_add -r automake bison curl gdbm git libtool libxml2
                    libxslt \
libyaml mysql55-server node-devel openssl readline sqlite3
                    sudo wget
```

rbenv

rbenv is the very first tool we need to install for setting up a *rails* installation. You may follow the documentation on *http://github.com/sstephenson/rbenv* (Listing 1).

About *RVM* – *rbenv* and *RVM* both serve the same purpose – that is, to ease the use of several ruby versions on a single computer. Both have their own particular advantages, drawbacks and philosophy. Anyhow, I didn't succeed in installing *RVM* on FreeBSD.

Listing 1. *rbenv installation*

```
$ git clone https://github.com/sstephenson/rbenv.git
                    ~/.rbenv
$ git clone https://github.com/sstephenson/ruby-build.
                    git \
  ~/.rbenv/plugins/ruby-build
$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >>
                    ~/.zshrc
$ echo 'eval "$(rbenv init -)"' >> ~/.zshrc
$ source ~/.zshrc
```

Ruby

As I am writing this, I recommend *ruby* versions 1.9.3 or 2.0.0, the latter being remarkably efficient.

```
$ rbenv install 2.0.0-p195
$ rbenv rehash
```

A *rehash* step is necessary each and every time a *rbenv* component (*ruby*, *gem*, *bundler*…) brings changes to the environment. To run our brand-new, freshly-installed *ruby* version by default:

```
$ rbenv global 2.0.0-p195
```

SQLite

The *sqlite3* options in the following command are *(sigh)* necessary for our *rails* installation to make use of it.

```
$ sudo gem install sqlite3 -- --with-sqlite3-dir=
   /usr/local --with-sqlite3-lib=/usr/local/lib
```

By default, *rails* will use s*qlite* in the development environment, and *mysql* in the production environment. Strangely enough, *mysql* will just work out-of-the box and will play with our *rails* installation pretty well – provided that you create the database and user your application will need (and that you actually start *mysql*).

Rails

```
$ gem install rails
$ rbenv rehash
```

That's just it! Now let's generate our project:

```
$ rails new bsdonrails
$ cd bsdonrails
$ rails s
```

Go visit *http://localhost:3000/* – it works! You may now want to set up a fixed *ruby* version in our working directory; do it like so:

```
$ rbenv local 2.0.0-p195
```

A *.ruby-version* file will then be added in the root directory of our project. For now, feel free to get a hold of the nice, OS-agnostic Ruby on Rails documentation. And start creating!

Playing with Rails

Rails provides us with generators, enabling us to create models, views and controllers that follow the MVC pattern. The scaffold generator created everything in one go:

Listing 2. *config/routes.rb*

```
Bsdonrails::Application.routes.draw do
  root to: 'posts#index'
  ressources: :posts
end
```

Listing 3. *rails console usage*

```
$ rails c
> p = Post.new(author: "Bob", content: "Hello, world!")
 => #<Post id: nil, author: "Bob", content: "Hello, world!", created_at: nil, updated_at: nil>
> p.save
> Post.all
 => [#<Post id: 1, author: "Bob", content: "Hello, world!", created_at: "2013-05-23 14:20:45", updated_at: "2013-05-
                   23 14:20:45">]
> Post.all.first.author
 => "Bob"
> p.author = "Alice"
> p.save
> Post.all.first.author
 => "Alice"
> exit
```

```
$ rails g scaffold Post author:string content:text
```

Numerous files are created: model, controller, view, Javascript, CSS, tests and database migration templates. Speaking of which, to migrate:

```
$ rake db:create # once per environment
$ rake db:migrate
```

Then go visit *http://localhost:3000/posts* – You may now manage your posts. How about that? To modify the landing page of your website so that the post listings get displayed instead, edit the *config/routes.rb* file like shown on Listing 2.

Then delete the "old" landing page:

```
$ rm public/index.html
```

Now *http://localhost:3000/* should display the posts' index.

Debug

Rails features a pleasing debug console like shown on Listing 3.

Production

Putting *rails* in production is as much a piece of cake as it was in the development environment (Listing 4).

`-d` makes server run as a daemon. Kill it with:

```
$ kill -9 `cat tmp/pids/server.pid`
```

Gems

More often than not, if you are looking for something that "has most likely already been developed", well, there probably is a *gem for* that. (Gems are sort of plugins for *rails*). To install them, modify the *Gemfile* in your project's root directory, before doing:

```
$ bundle install && rbenv rehash
```

Listing 4. *Production rails server*

```
$ RAILS_ENV=production rake db:create # once per
                    environment
$ RAILS_ENV=production rake db:migrate
$ bundle install && rbenv rehash
$ rake assets:clean assets:precompile
$ RAILS_ENV=production rails s -d
```

An average project may use several dozens of gems. A non-exhaustive, summed-up list is shown in Table 1. To use them, the wisest thing to do is to read their individual documentation on their GitHub page – sometimes modifying the *Gemfile* is not the only step to undertake.

Summary

We have set up a complete Ruby on Rails 3.2 workflow on FreeBSD 9.1. Feel free to let your creativity flow and make a project out of your idea, it is one of the best ways to learn. The official documentation is remarkably well done – make use of it!

Table 1. *Useful gems for your Gemfile*

acts_as_tenant	Multi-tenancy for SASS
bootstrap-sass font-awesome-rails	Twitter Bootstrap here I am!
cancan	Access Control
dalli	Memcached client
detect_timezone_rails	Localizes displayed datetime
devise devise-i18n devise-i18n-views	Authentication
omniauth omniauth-facebook omniauth-google-oauth2	Authentication with social networks (works well with devise)
faker	Generate fake data (name, address, e-mail, text…)
friendly_id	Use slug instead of id in URI: /categories/foobar
gravtastic	Gravatar
haml-rails	HAML is much better than ERB as templating language
jquery-rails modernizr-rails	Add your JS library through gems to update them easily
kaminari	Pagination
paperclip	Image upload and resizing through ImageMagick
paper_trail	Keep history of everything you want
private_pub	PubSub is awesome and simple
redcarpet pagedown-rails	Markdown and JS instant preview
rspec-rails capybara	Behavior-driven development. Test your views and JavaScript!
simple_form	Awesome forms helpers for your views
thin	Ruby Web server, just include it in Gemfile!
tire	Client for the Elasticsearch search engine/database.

On the Web

- *http://guides.rubyonrails.org/* – Official Rails guides,
- *http://railscasts.com/* – Rails screencasts by Ryan Bates, high quality inside,
- *https://github.com/plataformatec/devise#getting-started* – Devise gem for authentication, a good first gem to install.

Glossary

- Apache: HTTP server, to replace http://localhost:3000/ by http://what.i.want/,
- Bundler: gems manager, like Maven (Java) or Composer (PHP),
- Gem: Ruby program, library or Rails "plugin",
- MVC: Model, View, Controller pattern used in Rails.
- MySQL: relational database management system, to store data,
- Nginx: newer HTTP server,
- Rake: command-line software to execute named scripts,
- Rbenv: command-line software to install and switch between Ruby versions,
- Ruby: pure object-oriented programming language,
- Ruby on Rails: Ruby framework to build awesome websites,
- SQLite: another RDMS, used in development mode (easier to configure),
- Zsh: shell, like Bash but with more autocomplete stuff.

JULIEN GRILLOT
Formerly working for AF83 as a Ruby on Rails developer, Julien is a passionate Rails trainer at Paris. He contributes to open-source projects with the conviction that qualitative projects can be realized with little resources.
http://www.rubybb.com/ – Rails forum software (BSD),
http://event.chatchan.us/ – "Who's bringing what?" A Doodle-like tool.
julien.grillot@gmail.com

a d v e r i s e m e n t

m:tier

www.mtier.org
contact@mtier.org

- BSD development and consultancy
- Zabbix Monitoring
- Bacula enterprise backup
- BSD Thin Client
- Corporate BSD Desktop
- Solution management with Puppet
- and more ...

Creating PBI's with EasyPBI

The Push Button Installer (PBI) format is an easy-to-use package format for end-user applications that requires a specialized set of build instructions to create a PBI package. EasyPBI is designed to simplify the generation and use of these build instructions so that even non-technical users can quickly create and distribute applications as PBI packages.

The PBI package system is designed so that a single **.pbi* file contains not only the desired application but also all of the libraries and other dependencies required for the application to run. This necessitates that the PBI package process have two modes of operation: a "simple" mode that takes a local directory and packages it into a **.pbi* file without any modification and a "smart" mode that actually builds the application and its dependencies in a clean environment before packaging it all up. At the present time, the FreeBSD ports system is the only build framework that the PBI system can utilize to enable this "smart" build mode, but this could be extended in the future to support other build frameworks as well (such as the *pkgsrc* framework from NetBSD).

While the "simple" mode can be run with a one-line command, the "smart" mode requires quite a bit of specialized information to perform the build operations. This leads to the requirement of a directory of files that contains specialized build instructions for each individual PBI (hereafter referred to as a PBI "module"). This module must contain a configuration file (*pbi.conf*), and can optionally contain instructions for linking files from the PBI into the locally installed system hierarchy (*external-links*), and set up any XDG-compliant desktop/menu entries or mime types.

Module Generation with EasyPBI

Since the creation of these PBI modules can be both time-consuming as well as complicated for those unfamiliar with the PBI module format, EasyPBI was created by Jesse Smith and myself to provide a quick and easy way to generate PBI modules. Over time, additional features like running the PBI build process have been added to EasyPBI until now, with the 2.x series, EasyPBI has become a graphical front-end to the entire PBI build infrastructure, while still maintaining the simplicity and ease of use that defined the initial release.

The modules that EasyPBI can generate correspond to the two types of PBI builds that can be performed: "FreeBSD Port" modules are used by EasyPBI to run the "smart" build processes and the "Local Sources" modules are created specifically for EasyPBI to allow the user to run the "simple" build processes as well as include some of the extra PBI features previously restricted to the "smart" process. These two types of modules are available in the dialog (Figure 1) that appears when you click on the "New" button at the top of the EasyPBI window, and you can then give either the FreeBSD

Table 1. *PBI Configuration Options and Descriptions*

Name	Full Application Name
Version	Application Version (Local Sources Module Only)
Website	URL to the main application website
Author	Name of the application author(s)
Icon	Icon file for this application
Package Directory	Local directory to be packaged (Local Sources Module Only)
Main FreeBSD Port	FreeBSD port to be packaged (FreeBSD Port Module Only)
Port Build Options	(Un)Set configuration options for FreeBSD ports (FreeBSD Port Module Only)
Make Port Before	Additional port(s) to be built before the main application (FreeBSD Port Module Only)
Make Port After	Additional port(s) to be built after the main application (FreeBSD Port Module Only)
Requires Root Permissions	Check whether this application requires superuser permissions for installation/removal

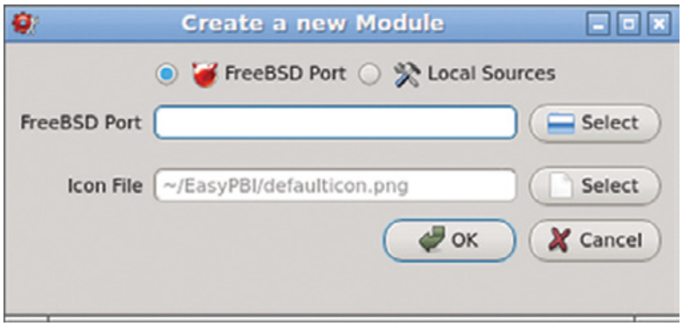


Figure 1. *New EasyPBI Module Dialog*

BSD Certification

The BSD Certification Group Inc. (BSDCG) is a non-profit organization committed to creating and maintaining a global certification standard for system administration on BSD based operating systems.

WHAT CERTIFICATIONS ARE AVAILABLE?

BSDA: Entry-level certification suited for candidates with a general Unix background and at least six months of experience with BSD systems.

BSDP: Advanced certification for senior system administrators with at least three years of experience on BSD systems. Successful BSDP candidates are able to demonstrate strong to expert skills in BSD Unix system administration.

WHERE CAN I GET CERTIFIED?

We're pleased to announce that after 7 months of negotiations and the work required to make the exam available in a computer based format, that the BSDA exam is now available at several hundred testing centers around the world. Paper based BSDA exams cost $75 USD. Computer based BSDA exams cost $150 USD. The price of the BSDP exams are yet to be determined.

Payments are made through our registration website:
https://register.bsdcertification.org//register/payment

WHERE CAN I GET MORE INFORMATION?

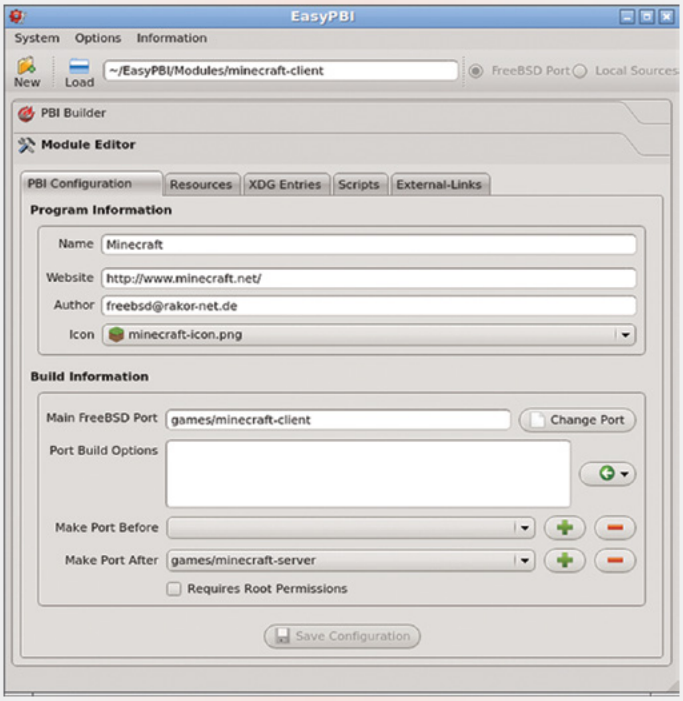
More information and links to our mailing lists, LinkedIn groups, and Facebook group are available at our website:
http://www.bsdcertification.org

Registration for upcoming exam events is available at our registration website:
https://register.bsdcertification.org//register/get-a-bsdcg-id

port (or local directory) that will be converted into a PBI. Also on this dialog, you may give the PNG file that you wish to use for the application icon, otherwise EasyPBI will assign a default icon that can be changed later as necessary (Figure 1).

Once a module has been created (or loaded), EasyPBI will display all of the options that are available within the configuration file for that module. If this is a new module for a FreeBSD port, EasyPBI will automatically read the port and set initial values for any of the configuration options possible. It is then possible to easily set or change the options. Just be sure to click the "save" button to keep your changes! While most of the configuration options are easily understandable, both they and their functions are listed here for reference (Table 1).

Once the PBI configuration is finished, your PBI is ready to be created. However, if the application is graphical in nature, it is probably a good idea to click over to the *XDG Entries* tab first (Figure 3). This tab allows you to create XDG compliant desktop/menu entries so that end-users can simply click on either a desktop icon or a listing in the application menu in order to start the application rather than resorting to the command line. Just for good measure, if the application has particular file extensions that it helps to run/manage, you can also associate a particular executable from the PBI with those file types. Whenever there is an arrow button next to a box that you can type into, EasyPBI will attempt to provide recommendations for that option. Simply click on the arrow button, and a menu list will appear that displays any of the solutions that EasyPBI could detect. Simply click



~/EasyPBI/Modules/minecraft-client [✓ FreeBSD Port] Local Sources

Name: Minecraft
Website: http://www.minecraft.net/
Author: freebsd@rakor-net.de
Icon: minecraft-icon.png

Main FreeBSD Port: games/minecraft-client
Port Build Options:
Make Port Before:
Make Port After: games/minecraft-server
☐ Requires Root Permissions

Figure 2. *PBI Configuration Editor*

on one of those solutions and it will either be added to or replace the current option value. These solutions are all found by reading through the FreeBSD port (if possible), so this is another situation where the information available in the build framework can simply be supplied through EasyPBI with a minimum of user effort at reading through or understanding the details of that build framework.

If the application that is to be packaged as a PBI is a command-line application (or if it is a local sources module), the *external-links* tab should also be checked before your PBI is completely ready (Figure 4). The *external-links* file provides a place to list all the files within the PBI that should be available to the end-user. For instance, any binaries that are listed here will have a special wrapper script created in the PBI and sym-linked onto the local system where the PBI is installed. This is also important for listing any *man* pages for the application or other files that are required in specific locations on the system. One thing to point out is that by default, if this is a module for a FreeBSD port, the main binaries listed in the port will automatically have external-links generated whether they are listed in this file or not. This ensures that at least the main application is available to be run on the user system after installation. If the current module is for packaging a local directory, you will need to take particular care to list all the application binaries here so that the user who installs this PBI will actually have the ability to run the application.

The last two tabs in the module editor are not used very often but are quite powerful when they are used. The *Resources* tab allows you to add additional files to the PBI. This is mainly used for adding the application icons, but it is possible to add other things such as default configuration files, binary wrapper scripts, and anything else that the application might be missing by default. The *Scripts* tab allows you to write your own custom scripts to be run during/after the PBI creation process (see Table 2 for a list of the possible scripts and when they are run). This allows you to perform custom build operations or modify the build process at any time.

One thing that is important to reiterate is that PBI's are built in a clean chroot environment without access to the host system. So if there are additional files that you need for a particular PBI build, you will need to fetch those files into the build environment by using one of these scripts. For advanced PBI scripting information, there is additional information as well as a list of predefined variables that can be found on the PC-BSD wiki page[1].

Building the PBI

EasyPBI also acts as a front-end to the PBI build process. Before doing this however, it might be a good idea to check the build settings in the EasyPBI Preferences.

The settings that are available are:

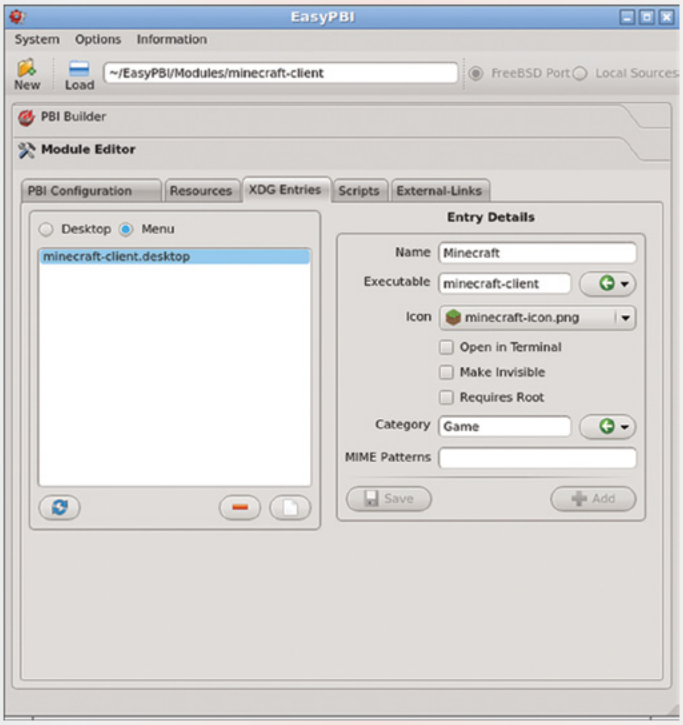
- set the directory where the completed PBI should be placed,
- digitally sign a PBI for tamper-evident distribution,
- use TMPFS to speed up builds (save temporary data to memory rather than hard disk),
- use package caching (re-use previously built packages for port dependencies).

By default the TMPFS and package caching options are turned off; however, I highly recommend enabling both of them due to the significant decrease in time it will take to build PBI's with EasyPBI. If you happen to see a build fail with *Out of Memory* errors though, you probably will need to turn off the TMPFS option because your system might not have enough memory to build that particular application (office suites are particularly large).

Once you have the PBI build settings configured, the *PBI Builder* within EasyPBI will give you an interface to the PBI build process. Only a single build process can be running at any given time, but once a build is started you can create or edit other modules while the PBI build is running. A PBI build requires root permissions (a prompt will appear before the build starts) as well as an active internet connection (to download files necessary to build applications). Once a build is started all of the log messages will be displayed in the EasyPBI interface in real-time, so you can keep an eye on it to see how it is proceeding. Should you need to cancel the build for some reason, there is a button on the side that will allow you to safely terminate the build process. Once the build is stopped (either cancelled or finished), EasyPBI also gives you the option to save that build log to a file should the need arise. This is especially important if your build ran into some kind of error and you need to seek assistance in resolving the issue. By saving the build log, you can provide the exact errors (usually at the end of the log) when asked to provide more details about the issue.

Other Resources

EasyPBI provides menu options for additional information about EasyPBI, FreeBSD ports, and the PBI module format. The EasyPBI option will open up a dialog that displays the EasyPBI licence (3-clause BSD) as well as the current version of EasyPBI and its development history. The FreeBSD ports option will try to open up a link to *www.freshports.org* in the default web browser. If you have a module loaded that uses a FreeBSD port, it will actually open the page that corresponds to that particular port. This is extremely useful if you want to quickly check what options are available for the port, what dependencies are required for the application to run, or other information that might be contained in the FreeBSD port. The last menu option will open up a link to the PC-BSD module builder's guide. This is useful if you are unsure what



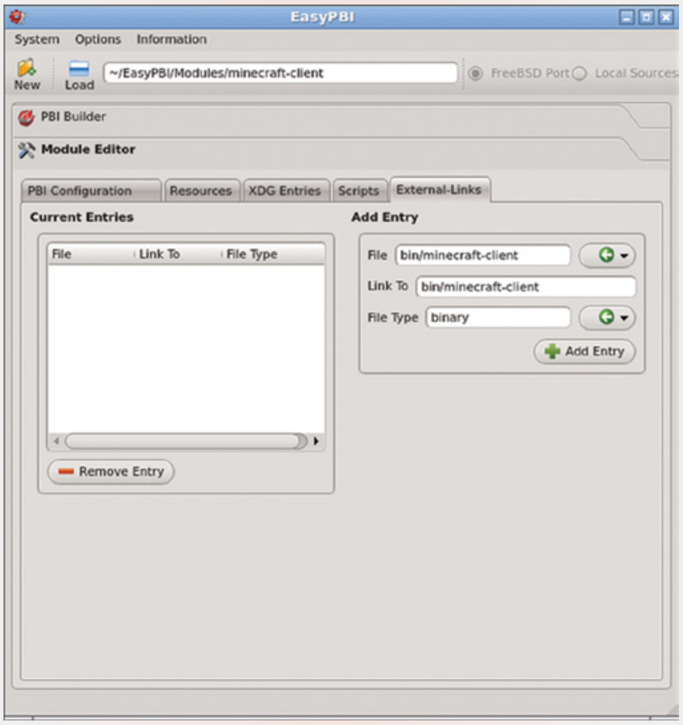
~/EasyPBI/Modules/minecraft-client

Desktop [✓ Menu]

minecraft-client.desktop

Name: Minecraft
Executable: minecraft-client
Icon: minecraft-icon.png
Open in Terminal
Make Invisible
Requires Root
Category: Game
MIME Patterns:

Figure 3. *XDG Menu Entry Editor*



~/EasyPBI/Modules/minecraft-client

File	Link To	File Type

File: bin/minecraft-client
Link To: bin/minecraft-client
File Type: binary

Figure 4. *External-Links Editor*

a particular option is used for or if you want to learn more about the PBI module specifications for advanced configurations.

Submission of a Module for addition to the PC-BSD repository

Once you have a working PBI module, I would recommend submitting it for inclusion into a PBI repository so that others can use the resulting PBI as well. Generally, a PBI repository will run a PBI building daemon that will automatically create and update any PBI's that it makes available. To do this, they only need the set of build instructions (the PBI module) which can then be added to the module tree that the daemon oversees. In order to assist in the submission of PBI modules to a repo, EasyPBI provides a menu option to compress a copy of the currently loaded module into a small **.tar.gz* file for transport via email or other methods.

Table 2. *Available Scripts and Runtimes*

pre-portmake.sh	Start of the build process
post-portmake.sh	After building the listed ports
pre-pbicreate.sh	After formatting the PBI dir, but before it is packaged as a PBI
pre-install.sh	Before the PBI is installed on a system
post-install.sh	After the PBI is installed on a system
pre-remove.sh	Before the PBI is removed from a system

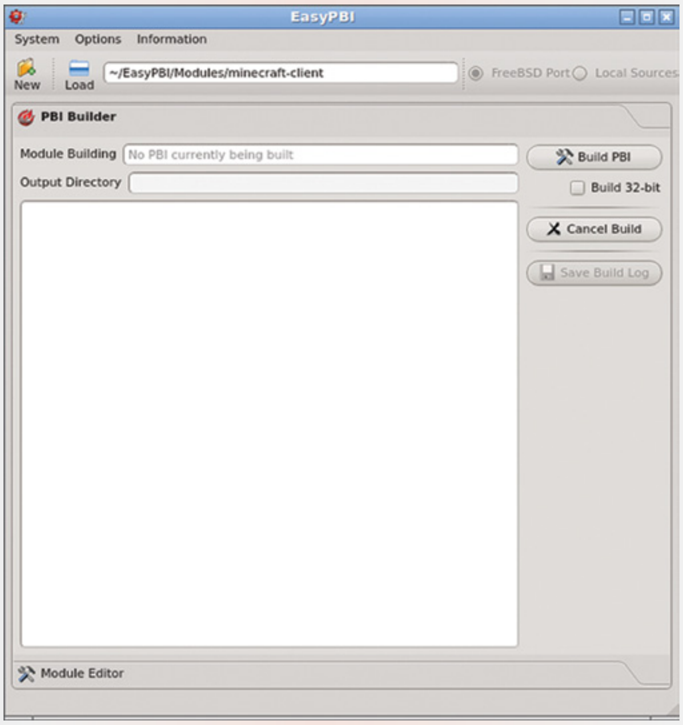


Figure 5. *PBI Builder Interface*

On the Web

- *http://wiki.pcbsd.org/index.php/PBI_Module_Builder_Guide* – PBI Module Builder's Guide
- *http://forums.pcbsd.org* – PC-BSD forums (with PBI developers sub-forums)
- *http://lists.pcbsd.org/mailman/listinfo* – PC-BSD mailing lists
- *https://github.com/pcbsd/pbi* – PC-BSD PBI repository on GitHub

The PC-BSD project currently provides a repository with over eleven hundred PBI's available, and we are always looking for more! There are two main methods by which modules may be submitted to the PC-BSD repository. First, you can send the packaged module in an email to the PC-BSD PBI developers mailing list [3]. One of the repo managers will then check that module for accuracy and make any small adjustments that might be needed (usually just fixing the icons or desktop/menu entries) and then add it to the repository. The other method is fairly new, but the PC-BSD repository is now available on *GitHub* [4] and as such you can, fork the repo, make your changes, and then send us a pull request to have your changes checked and merged back into the main branch.

Summary

EasyPBI is a complete graphical front-end to the PBI creation process that makes the creation of PBI packages simple for all users. By using EasyPBI, you not only get a streamlined process with automatic form generation and simplifications, but you also retain the power of creating PBI modules by hand. In addition, EasyPBI provides the ability to package local directories into the PBI format while retaining the ability to add XDG-compliant desktop/menu entries or mime types to the new PBI. All of this adds up to a program that is the recommended method for generating new PBI's for individual or commercial use.

KEN MOORE

Ken Moore co-created EasyPBI with Jesse Smith in 2011 and took over full development of it for the PC-BSD project in 2012. He lives in Tennessee with his wife and two sons and is always looking for ways to make computers simpler, but no less powerful, for the average user. He is currently employed by iXsystems to work on the PC-BSD Project as both a developer and as the manager for the PC-BSD PBI repository. He can be reached at: ken@pcbsd.org.

SPRING FUNDRAISING DRIVE

Support FreeBSD by donating

Your donations have helped make FreeBSD the best OS available! By investing in the services provided by The FreeBSD Foundation you have helped us fund projects to keep FreeBSD a high-performance, secure, and stable OS.

What will the Foundation accomplish with your donation in 2013?

- **Software development projects** for FreeBSD: **$600,000.**
- Paid staff time supporting **Release Engineering and Security teams.**
- **Grow staff: Five technical staff members** by year-end.
- **Provide support for BSD conferences around the globe,** in Europe, Japan, Canada, and the USA.
- **Hardware** to maintain and improve FreeBSD project infrastructure: **$130,000.**
- **FreeBSD community growth** through marketing and outreach to users and businesses.
- **Legal services and counsel protecting the FreeBSD trademarks.**

FreeBSD is internationally recognized as an innovative leader in providing a high-performance, secure, and stable operating system. Our mission is to continue and increase our support and funding to keep FreeBSD at the forefront of operating system technology. But, we can't do this without your help!

Last year with your generosity, we raised over $770,000. This year we will invest $1,000,000 to support and promote FreeBSD.

We have kicked off the new year with three newly funded projects, and are actively soliciting additional project proposals.

Please support the Foundation during our Spring Fundraising Drive, and help us raise $100,000 from 1000 donors between April 15th and May 30th.

We need your help.
We can't do this without you...

Make your donation today. Go to:
www.freebsdfoundation.org/donate

The FreeBSD FOUNDATION

Then talk to your employer about matching your gift— or making their own donation.

Find out more, visit:

www.freebsdfoundation.org

Manage your Ruby Versions Under FreeBSD

Ruby Version Manager is a great tool to manage several Ruby binaries without dependency breaks. The examples from this article have been tested under FreeBSD 9.1 with bash.

What you will learn…

- You'll learn to install and use this awesome project for your web, or another type of developed projects with Ruby programming language with FreeBSD

What you should know…

- Basic Shell skills.
- Very basic Ruby ecosystem understanding.
- Nginx virtualhost configuration.

Developed by Wayne E. Seguin, RVM allows a system administrator or Ruby developer to install several versions of their favorite scripting language, Ruby. For each Ruby version, you will find the rubygems utility to install, update, remove, or build your needed gems.

RVM setup

You can deploy RVM as root or as a simple user. For your introduction with RVM, test with a standard user (single user install). Root's install is for multi-user usage.

Here we go, we need some binary to set up RVM:

```
 tib@cendrillon$ for name in {bash,awk,sed,grep,ls,cp,
tar,curl,gunzip,bunzip2,git,svn} ; do which $name ;  done
```

After getting the requirements, install RVM:

```
tib@cendrillon$ \curl -L https://get.rvm.io | bash
```

Load RVM (you can open a new session too):

```
tib@cendrillon$ source ~/.rvm/scripts/rvm
```

After this step, to test your setup, you must see "rvm is a shell function", and type:

Listing 1. *rvm list known*

```
tib@cendrillon$ rvm list known
# MRI Rubies
[ruby-]1.8.6[-p420]
[ruby-]1.8.7[-p371]
[ruby-]1.9.1[-p431]
[ruby-]1.9.2[-p320]
[ruby-]1.9.3-p125
[ruby-]1.9.3-p194
[ruby-]1.9.3-p286
[ruby-]1.9.3-p327
[ruby-]1.9.3-p362
[ruby-]1.9.3-p374
[ruby-]1.9.3-p385
[ruby-]1.9.3-p392
[ruby-]1.9.3[-p429]
[ruby-]1.9.3-head
[ruby-]2.0.0-rc1
[ruby-]2.0.0-rc2
[ruby-]2.0.0-p0
[ruby-]2.0.0[-p195]
ruby-head
…
```

```
tib@cendrillon$ type rvm | head -n 1
rvm is a shell function
```

That's it. RVM is set up.

Note: If you get "rvm is not a function", you need to configure your shell as login shell.

Listing 2. *rvm install*

```
tib@cendrillon$ rvm install 1.8.7
Searching for binary rubies, this might take some time.
No binary rubies available for: freebsd/9.1-RELEASE-p3/x86_64/ruby-1.8.7-p371.
Continuing with compilation. Please read 'rvm mount' to get more information on binary rubies.
Installing Ruby from source to: /home/tib/.rvm/rubies/ruby-1.8.7-p371, this may take a while depending on your
                    cpu(s)...
ruby-1.8.7-p371 - #downloading ruby-1.8.7-p371, this may take a while depending on your connection...
ruby-1.8.7-p371 - #extracted to /home/tib/.rvm/src/ruby-1.8.7-p371 (already extracted)
Patch stdout-rouge-fix was already applied.
Patch no_sslv2 was already applied.
#configuring
#compiling
#installing
Retrieving rubygems-1.8.25
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  371k  100  371k    0     0  1542k      0 --:--:-- --:--:-- --:--:-- 3831k
Extracting rubygems-1.8.25 ...
Removing old Rubygems files...
Installing rubygems-1.8.25 for ruby-1.8.7-p371
Installation of rubygems completed successfully.
Saving wrappers to '/home/tib/.rvm/wrappers/ruby-1.8.7-p371'.......

ruby-1.8.7-p371 - #adjusting #shebangs for (gem irb erb ri rdoc testrb rake).
ruby-1.8.7-p371 - #importing default gemsets, this may take time......................
Install of ruby-1.8.7-p371 - #complete
Please be aware that you just installed a ruby that requires 2 patches just to be compiled on up to date linux
                    system.
This may have known and unaccounted for security vulnerabilities.
Please consider upgrading to ruby-2.0.0-p195 which will have all of the latest security patches.
```

Listing 3. *rvm list*

```
tib@cendrillon$  rvm list

rvm rubies

   ruby-1.8.7-p371 [ x86_64 ]
   ruby-1.9.3-p392 [ x86_64 ]

# Default ruby not set. Try 'rvm alias create default <ruby>'.

# => - current
# =* - current && default
#  * - default
```

Listing 4. *Rakefile*

```
require 'rake/testtask'

Rake::TestTask.new do |t|
  t.libs << "test"
  t.test_files = FileList['test.rb']
  t.verbose = true
end
```

Listing 5. *example.rb*

```
require "test/unit"

class TestBsdMag < Test::Unit::TestCase

  def test_hash
    h = {"alice", "bob"}
    assert_equal({"alice" => "bob"}, h)
  end

end
```

Listing 6. *rake test*

```
tib@cendrillon$ rvm use 1.8.7 && rake test
Loaded suite /home/tib/.rvm/gems/ruby-1.8.7-p371@global/
                    gems/rake-10.0.4/lib/rake/rake_test_
                    loader
Started
.
Finished in 0.000923 seconds.

1 tests, 1 assertions, 0 failures, 0 errors
```

Listing 7. *rake test*

```
tib@cendrillon$ rvm use 1.9.3-392 && rake test
Using /home/tib/.rvm/gems/ruby-1.9.3-p392
/home/tib/.rvm/rubies/ruby-1.9.3-p392/bin/ruby
                    -I"lib:test" -I"/home/tib.rvm/gems/
                    ruby-1.9.3-p392@global/gems/rake-
                    10.0.4/lib" "/home/tib.rvm/gems/ruby-
                    1.9.3-p392@global/gems/rake-10.0.4/
                    lib/rake/rake_test_loader.rb" "test.
                    rb"
/home/tib.rvm/rubies/ruby-1.9.3-p392/lib/ruby/site_
                    ruby/1.9.1/rubygems/custom_require.
                    rb:36:in `require': /home/tib/test.
                    rb:6: syntax error, unexpected ',',
                    expecting tASSOC (SyntaxError)
    h = {"alice", "bob"}
                 ^
/home/tib/test.rb:6: syntax error, unexpected '}',
                    expecting keyword_end
```

Listing 8. *test.rb*

```
require "test/unit"

class TestBsdMag < Test::Unit::TestCase

  def test_hash
    h = {"alice" => "bob"}
    assert_equal({"alice" => "bob"}, h)
  end

end
```

Listing 9. *rake test*

```
tib@cendrillon$ rvm use 1.8.7 && rake test
Loaded suite /home/tib/.rvm/gems/ruby-1.8.7-p371@global/
                    gems/rake-10.0.4/lib/rake/rake_test_
                    loader
Started
.
Finished in 0.000923 seconds.

1 tests, 1 assertions, 0 failures, 0 errors
```

Ruby switching, install gems

Before using this new tool, ensure that you have all the needed dependencies and build the port of Ruby to have all the stuff needed in the future (faster method):

```
root@cendrillon# cd /usr/ports/lang/ruby19 && make install
        clean && cd -
```

Listing 10. *rake test*

```
tib@cendrillon$ rvm use 1.9.3-p392 && rake test
/home/tib.rvm/rubies/ruby-1.9.3-p392/bin/ruby -I"lib:test" -I"/home/tib.rvm/gems/ruby-1.9.3-p392@global/gems/rake-
                    10.0.4/lib" "/home/tib.rvm/gems/ruby-1.9.3-p392@global/gems/rake-10.0.4/lib/rake/rake_test_
                    loader.rb" "test.rb"
Run options:

# Running tests:

Finished tests in 0.001735s, 576.4267 tests/s, 576.4267 assertions/s.

1 tests, 1 assertions, 0 failures, 0 errors, 0 skips
```

Listing 11. *adduser*

```
root@cendrillon# adduser
Username: test_unicorn
Full name: Test Unicorn
Uid (Leave empty for default):
Login group [test_unicorn]:
Login group is test_unicorn. Invite test_unicorn into other groups? []:
Login class [default]:
Shell (sh csh tcsh zsh rzsh git-shell bash rbash nologin) [sh]: bash
Home directory [/home/test_unicorn]: /usr/local/www/test_unicorn
Home directory permissions (Leave empty for default):
Use password-based authentication? [yes]: yes
Use an empty password? (yes/no) [no]:
Use a random password? (yes/no) [no]:
Enter password:
Enter password again:
Lock out the account after creation? [no]:
Username    : test_unicorn
Password    : *****
Full Name   : Test Unicorn
Uid         : 5003
Class       :
Groups      : test_unicorn
Home        : /usr/local/www/test_unicorn
Home Mode   :
Shell       : /usr/local/bin/bash
Locked      : no
OK? (yes/no): yes
adduser: INFO: Successfully added (test_unicorn) to the user database.
Add another user? (yes/no): no
Goodbye!
```

You have RVM on your system now and you are going to install Rubies. But which Rubies? We need to list what Rubies are available (Listing 1).

I didn't display the full result here, but this command shows available rubies version with different interpreters (MRI here). Before installing rubies with RVM, to avoid problem with RVM's autolibs feature, disable it:

```
tib@cendrillon$ rvm autolibs 0
```

If you don't disable it, the following command won't work. It will be looking up for compiling dependencies (optimized for GNU/Linux and MacOS X system) for a very long time. We chose Ruby 1.8.7 (deprecated now) and Ruby 1.9.3-p392 for our future tests (Listing 2) and:

```
tib@cendrillon$ rvm install 1.9.3-p392
[… same things like above ...]
```

So we have two rubies (Listing 3). We want to use Ruby 1.9 as default:

```
tib@cendrillon$ rvm use 1.9.3 --default
Using /home/tib/.rvm/gems/ruby-1.9.3-p392
```

Where is my current Ruby binary?

```
tib@cendrillon$ which ruby
/home/tib/.rvm/rubies/ruby-1.9.3-p392/bin/ruby
```

You want to switch to your system Ruby, installed via ports tree:

```
tib@cendrillon$ rvm use system
Now using system ruby.
```

Ensure that's ok:

```
tib@cendrillon$ which ruby
/usr/local/bin/ruby
```

You have two rubies and RVM installed. It's time to learn some use case examples.

Testing use case

You are an awesome developer who has been hired to develop cross compatibility between Ruby 1.8.7 and Ruby 1.9.3 of an existing Ruby program currently running in Ruby 1.8.7. Your application has some unit tests. For this

Listing 12. */usr/local/www/test_unicorn/bob/config/unicorn.rb*

```
worker_processes 2
working_directory "/usr/local/www/test_unicorn/bob/"

preload_app true

timeout 30

listen "/usr/local/www/test_unicorn/bob/tmp/sockets/unicorn.sock", :backlog => 64

pid "/usr/local/www/test_unicorn/bob/tmp/pids/unicorn.pid"

stderr_path "/usr/local/www/test_unicorn/bob/log/unicorn.stderr.log"
stdout_path "/usr/local/www/test_unicorn/bob/unicorn.stdout.log"

before_fork do |server, worker|
  defined?(ActiveRecord::Base) and
    ActiveRecord::Base.connection.disconnect!
end

after_fork do |server, worker|
  defined?(ActiveRecord::Base) and
    ActiveRecord::Base.establish_connection
end
```

quick and dirty example, my tests are represented by two assertions with the test unit.

You will need two files with related Ruby code (Listing 4 and 5). Test our mini test suite (Listing 6). And now, with 1.9.3 (Listing 7). This code doesn't work with Ruby 1.9.3 because simple hash allocation has been removed. We need patch test.rb following Listing 8. Relaunch tests (Listing 9). That's still OK with 1.8.7 (Listing 10). And it works with 1.9.3! Regression tests, OK.

Imagine if you used the old way to construct your hashes in your current app. Without tests, you would be screwed. Unit tests and RVM allow you to test your app faster.

When you develop an application, it can be useful to make tests (unit, functional, etc.) to avoid bugs and save time. Test your app with different versions of Ruby – very – easily with RVM.

Further application

Now, a web deployment example for sysadmins. We are going to set up a rails application example with unicorn (used by Github), Nginx, and RVM.

You need to install Nginx (passenger isn't needed):

```
root@cendrillon# cd /usr/ports/www/nginx && make install
                    clean && cd -
```

Create a user for our test (Listing 11).

Become `test_unicorn`:

```
root@cendrillon# su - test_unicorn
```

Now to train yourself, install RVM and rubies for `test _ unicorn. =]`.

Listing 13. *config*

```
upstream unicorn_test_server {
   server unix:/usr/local/www/test_unicorn/bob/tmp/sockets/unicorn.sock
   fail_timeout=0;
}

server {
        listen   80;
        server_name  a.fraking.domain;
        root /usr/local/www/test_unicorn/bob/public;

        location / {
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
          proxy_set_header Host $http_host;
          proxy_redirect off;

          # If you don't find the filename in the static files
          # Then request it from the unicorn server
          if (!-f $request_filename) {
            proxy_pass http://unicorn_test_server;
            break;
          }
        }

        error_page 500 502 503 504 /500.html;
        location = /500.html {
          root /usr/local/www/test_unicorn/bob/public;
        }

}
```

After these steps are completed (I setup 1.9.3-p392), install Rails gem:

```
test_unicorn@cendrillon$ gem install rails
```

You will need sqlite3:

```
root@cendrillon# cd /usr/ports/databases/sqlite3 && make
                    install clean && cd -
```

Create a new application called bob (wait during bundle install, don't ^C):

```
test_unicorn@cendrillon$ rails new bob
```

Install unicorn:

```
test_unicorn@cendrillon$ gem install unicorn
```

Fill the unicorn configuration file (Listing 12). Create a new vhost with this config (Listing 13).

Run unicorn:

```
test_unicorn@cendrillon$ cd /usr/local/www/test_unicorn/bob
test_unicorn@cendrillon$ mkdir -p tmp/pids
test_unicorn@cendrillon$ mkdir -p tmp/sockets
test_unicorn@cendrillon$ unicorn -c /usr/local/www/
   test_unicorn/bob/config/unicorn.rb -E production -D
```

Restart Nginx on your new virtual host:

```
root@cendrillon# service nginx restart
```

Connect to your virtual host configured url. You should see that page:

```
"The page you were looking for doesn't exist."
```

That's the Rails framework 404 error. Check log/production.log! You need to configure the app, set the database, add a new controller, add a base route, well, code a web app! Unicorn works!

Yeah, it works. Stop unicorn and observe the difference.

Yeah, that's it. It's not a review of Rails development (Rails can need some big dependencies, like mysql, etc.), nor a full review of Nginx configuration. But, if you're running applications in production with Rails (or other rack based applications), you can think about useful tricks to customize your actual configuration with, for example, "per user virtual host configurations" or "per rails applications" configurations, adding gemsets for each app, try thin, try passenger, try Mongrel, etc. This example isn't production ready; its purpose is to give you a basic understanding to continue the adventure.

Normally, I don't use FreeBSD (except for this article). Can RVM work with all *BSD systems?

In theory, yes. But I haven't tested it. So, I would recommend testing RVM for another BSD system before thinking about production environment deployment.

If you want an RVM alternative, you can test rbenv (see link section). At work, I use OpenBSD every day as my desktop station and I work with rbenv (just out of curiosity). I didn't try to use RVM or rbenv under NetBSD or DragonflyBSD; it's still on my todo list.

Summary

As you saw during this quick introduction, RVM can be an amazing tool both for developers and for sysadmins. I hope this article gives you some great ideas to run more awesome Ruby applications in production!

On the web

- *https://rvm.io/* – RVM website
- *https://github.com/wayneeseguin/rvm* – RVM Github
- *http://rbenv.org/* – rbenv website
- *https://github.com/sstephenson/rbenv* – rbenv Github
- *http://www.ruby-lang.org/en/* – Ruby language website
- *http://guides.rubyonrails.org/getting_started.html* – RoR Getting started tutorial
- *https://github.com/blog/517-unicorn* – Unicorn description

THIBAUT DELOFFRE

Thibaut Deloffre discovered BSD systems during his studies and continues to use them every day as a sysadmin and developer (Ruby ;)). He worked for LINAGORA, a French company advocating open source softwares in France, as a LAN administrator. Thibaut works now as a system and network integrator for a telecom operator in France.

Great Specials

On FreeBSD® & PC-BSD® Merchandise

Give us a call & ask about our
SOFTWARE BUNDLES
1.925.240.6652

$39.95
FreeBSD 9.1 Jewel Case CD Set
or FreeBSD 9.1 DVD

$29.95
PC-BSD 9.1 DVD

Save a BUNDLE!

$49.95
The PC-BSD 9.0 Users Handbook
PC-BSD 9.1 DVD

$99.95
The FreeBSD CD or DVD Bundle

Inside each CD/DVD Bundle, you'll find:
FreeBSD Handbook, 3rd Edition
Users Guide FreeBSD Handbook, 3rd Edition, Admin Guide
FreeBSD 9.1 CD or DVD set
FreeBSD Toolkit DVD

Stylish Dress Attire
Look Your Professional Best

Comfy Apparel
Stay Warm in Zip Ups & Pullovers

T-Shirts
Lots of Styles to Choose From

FreeBSD 9.1 Jewel Case CD/DVD	$39.95

CD Set Contains:

Disc 1	Installation Boot LiveCD (i386)
Disc 2	Essential Packages Xorg (i386)
Disc 3	Essential Packages, GNOME2 (i386)
Disc 4	Essential Packages (i386)

FreeBSD 9.0 CD	$39.95
FreeBSD 9.0 DVD	$39.95

FreeBSD Subscriptions

Save time and $$$ by subscribing to regular updates of FreeBSD

FreeBSD Subscription, start with CD 9.1	$29.95
FreeBSD Subscription, start with DVD 9.1	$29.95
FreeBSD Subscription, start with CD 9.0	$29.95
FreeBSD Subscription, start with DVD 9.0	$29.95

PC-BSD 9.1 DVD (Isotope Edition)

PC-BSD 9.1 DVD	$29.95
PC-BSD Subscription	$19.95

The FreeBSD Handbook

The FreeBSD Handbook, Volume 1 (User Guide)	$39.95
The FreeBSD Handbook, Volume 2 (Admin Guide)	$39.95

The FreeBSD Handbook Specials

The FreeBSD Handbook, Volume 2 (Both Volumes)	$59.95
The FreeBSD Handbook, Both Volumes & FreeBSD 9.1	$79.95

PC-BSD 9.0 Users Handbook	$24.95
BSD Magazine	$11.99
The FreeBSD Toolkit DVD	$39.95
FreeBSD Mousepad	$10.00
FreeBSD & PCBSD Caps	$20.00
BSD Daemon Horns	$2.00

Bundle Specials!
Save $$$

Just Plain Fun
Mousepads & Novelty Horns

BSD Magazine
Available Monthly

FreeBSD Mall

For even MORE items visit our website today!
www.FreeBSDMall.com

Keep OpenBSD Customers Satisfied

For a long time there was nothing like security updates for OpenBSD packages. Now M:Tier company has introduced a new long-term support and update service for OpenBSD.

What you will learn…

- What are binpatches
- What are -stable package updates
- How to start using update service on OpenBSD

What you should know…

- OpenBSD is issued every 6 months and does not offer any package updates. Core system updates are in the form of source patches, and every release is supported only for 6 months, then users are advised to upgrade to the next release.
- During the development of a new release, the unfinished source called -current can be tested together with alpha packages from the snapshots directory. A user can download the -current system, install packages and upgrade both the system and packages every time a binary upgrade is available. The upgrade requires in fact a reinstallation of the whole core system every time.

OpenBSD and -stable source patches. M:Tier's binpatches provide security updates for OpenBSD in the form of binary patches which can be installed as regular packages.

For a long time, OpenBSD has been doing two releases per year. Security updates are being committed to the source tree, but no new installation sets are built with these updates. Up until now, OpenBSD could not measure up to many popular Linux distributions when it came to security updates for previous releases.

The M:Tier team is working to provide all OpenBSD users free support with the introduction of binpatches and updated packages (from ports) with security fixes. This started in the form of binpatches for the base system with OpenBSD 5.2 and it has now been extended to include updated packages for OpenBSD 5.3.

Binpatches and what's the advantage?

Normally source becomes updated and each person can make a binary on their own on each machine and install updated system parts. This is where M:Tier stood up and provided binary patches that offer an easy way for users to apply the security fixes to their *-stable* installation.

This is how it works, imagine you take care of three servers and run one instance of OpenBSD on your laptop for testing reasons. Now you learn that a new security hole or memory leak was found in this or that part of the system. At this moment if you want to apply the fixing patch you have to:

- download system sources
- apply the patch
- compile the whole core system
- apply the update
- and you have to do that on every single machine you maintain.

In case you use *binpatchNG* (a framework for creating binary patches for OpenBSD on all platforms in a semi-automatic way, developed by M:Tier), you can generate a binary patch on one machine and then apply it on the other machines.

And if you decide to use M:Tiers binpatch repository, you really only need to download and apply the patch on each of your machines. You don't have to download sources or compile anything. You in fact just install a binpatch in the form of a package. The saved time and work is multiplied by the number of machines you have to maintain, of course.

What are -stable package updates?

OpenBSD continuously builds packages from the ports tree for *-current*, but it only builds the packages for a release once and does not provide any upgrades or security updates. This is where the stable package updates come in; they are built and released whenever a security update or fix has been committed to the ports tree. M:Tier developers apply the security and stability fixes to the tree and provide freshly-updated packages in their repository. The repository does not contain any additional packages or newer versions of packages, just security and stability fixes.

How to use M:Tier's stable updates?

Using M:Tier's *-stable* repository is quite easy. It requires adding the M:Tier repository to your PKG_PATH so the *pkg_** tools know where to look for updates. Secondly, it requires importing the SSL certificate with which the packages have been signed. This latter feature has been added so that users can always validate that the packages they install are really built by M:Tier.

1) Install the M:Tier certificate

Retrieve the certificate from *https://stable.mtier.org* and install it into:

```
/etc/ssl/pkgca.pem
```

2) Update your PKG_PATH

Please update your `$PKG_PATH` environment variable to:

```
PKG_PATH=https://stable.mtier.org/updates/$(uname
                        -r)/$(arch -s):${PKG_PATH}
```

a d v e r t i s e m e n t

For example:

```
PKG_PATH=http://ftp.fr.openbsd.org/pub/OpenBSD/$(uname
-r)/packages/$(arch  -s)  PKG_PATH=https://stable.mtier.
org/updates/$(uname  -r)/$(arch  -s):${PKG_PATH}  export
PKG_PATH
```

Finally you can check the variable, and you should get something similar to this:

```
$ env |grep PKG_PATH
PKG_PATH=https://stable.mtier.org/updates/5.3/i386:http://
ftp.fr.openbsd.cs.fau.de/pub/OpenBSD/5.3/packages/i386
```

When you adjust your system accordingly, you are ready to use the binpatches and packages update directory.

Installing them, there is a difference

With packages, which are ready for 5.3 and later, it works the same way like it works with packages in the current OpenBSD snapshots. So the only thing you have to do is to run this as a root as shown on Figure 1.

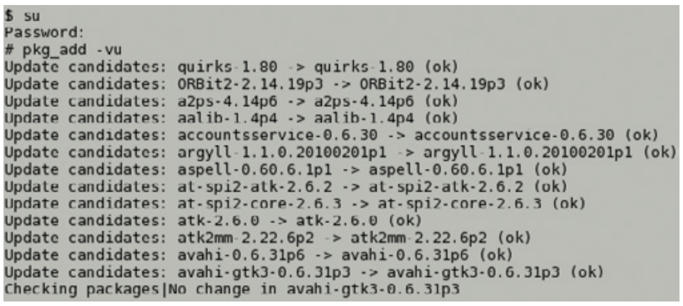
Then you can go and have your coffee. `Pkg_add` will check your currently installed packages for any available updates, and installs them if they are found.

If for some reason you only want to update a single package, say your PostgreSQL database server, you can run:

```
pkg_add -u postgresql-server
```

Binpatches, which have been working since 5.2 already, have to be installed manually, as they update the basic part of the system and `pkg_add` does not recognize them as updates for now. However binpatches come in the form of packages too, so you just use the name of the binpatch, for example a kernel:

```
pkg_add binpatch53-amd64-kernel-1.0.tgz
```



```
$ su
Password:
# pkg_add -vu
Update candidates: quirks-1.80 -> quirks-1.80 (ok)
Update candidates: ORBit2-2.14.19p3 -> ORBit2-2.14.19p3 (ok)
Update candidates: a2ps-4.14p6 -> a2ps-4.14p6 (ok)
Update candidates: aalib-1.4p4 -> aalib-1.4p4 (ok)
Update candidates: accountsservice-0.6.30 -> accountsservice-0.6.30 (ok)
Update candidates: argyll-1.1.0.20100201p1 -> argyll-1.1.0.20100201p1 (ok)
Update candidates: aspell-0.60.6.1p1 -> aspell-0.60.6.1p1 (ok)
Update candidates: at-spi2-atk-2.6.2 -> at-spi2-atk-2.6.2 (ok)
Update candidates: at-spi2-core-2.6.3 -> at-spi2-core-2.6.3 (ok)
Update candidates: atk-2.6.0 -> atk-2.6.0 (ok)
Update candidates: atk2mm-2.22.6p2 -> atk2mm-2.22.6p2 (ok)
Update candidates: avahi-0.6.31p6 -> avahi-0.6.31p6 (ok)
Update candidates: avahi-gtk3-0.6.31p3 -> avahi-gtk3-0.6.31p3 (ok)
Checking packages|No change in avahi-gtk3-0.6.31p3
```

Figure 1. *pkg_add -vu*

On the Web

- *https://stable.mtier.org/*
- *http://www.undeadly.org/cgi?action=article&sid=20130509120042*
- *http://www.undeadly.org/cgi?action=article&sid=20110420080633*
- *http://www.openports.se/*
- *http://www.openbsd.org/*
- *http://www.openunix.eu/*

and the package is downloaded and applied to the system. For ease of use, the developers have added an rsync access, then finding out names and applying patches is easier, use the following command, and you can inspect and install all the latest locally:

```
rsync -av --progress rsync://stable.mtier.org/OpenBSD-
                    Stable/ \  /my/favourite/folder/
```

Conclusion

M:Tier binpatches and *-stable* package updates really change the world of OpenBSD. The good thing is not only that you can keep your system updated just as many Linux users get it, but the support is longer than it used to be. Now you do not have to upgrade or reinstall your OpenBSD every half a year, but you can stay on stable and keep your system updated for one whole year. Also the fact that some core OpenBSD developers are part of M:Tier ensures the development of updates is in line with the development of the OpenBSD system.

Final word

Finally I would very much like to thank M:Tier developers for their job and for being helpful and friendly even in moments when the writer of this article did not keep his wits about him.

PETR TOPIARZ

The author has been administering BSD web/mail/file servers and Linux desktops in three small Prague-based companies for the last eight years, and has contributed to BSDMag since its first issues.

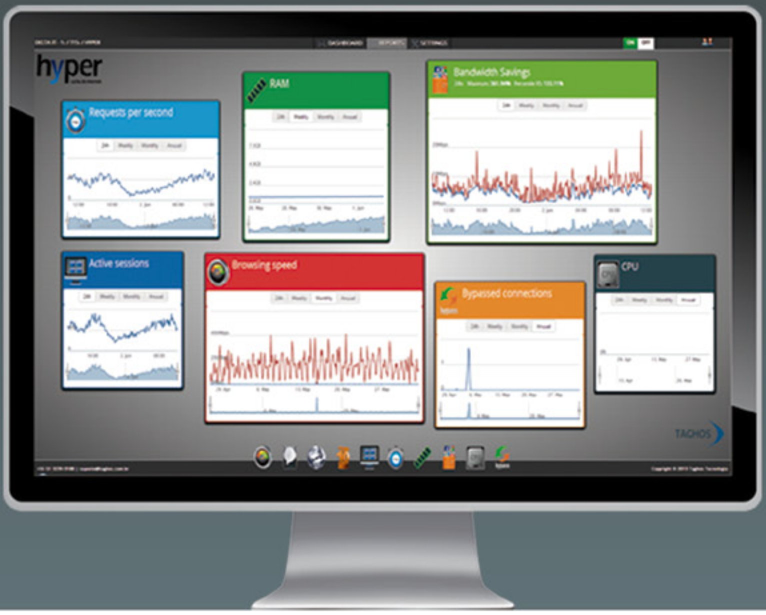
Faster.
Better.
Reliable.

Trusted by over 500 ISPs worldwide.

Hyper is the first multimedia cache fully developed in Brazil, by Taghos.
With Hyper, ISPs can save on network bandwidth while increasing content-delivery speeds, resulting in end-customer satisfaction.

Features:

- 24x7x365 always-on support
- Active monitoring
- Automatic updates
- Appliance or license
- Easy deployment
- Configuration and reports via web interface



hyper

Remote Install
Using your hardware

Model	Traffic	RAM	Cache	SSD
T15	Up to 15 Mbps	8 GB	1x 1 TB	-
T50	Up to 50 Mbps	8 GB	2x 1 TB	-
T100	Up to 100 Mbps	8 GB	2x 1 TB	1x 160 GB
T150	Up to 150Mbps	16 GB	3x 2 TB	1x 160 GB
T300	Up to 300 Mbps	16 GB	5x 2 TB	1x 240 GB
T500	Up to 500 Mbps	32 GB	7x 2 TB	1x 480 GB
T1000	Up to 1 Gbps	64 GB	10x 1 TB	1x 480 GB
T2000	Up to 2 Gbps	96 GB	24x 1 TB	3x 480 GB
T3000	Up to 3 Gbps	128 GB	32x 1 TB	5x 480 GB

Visit us at **www.taghos.com** and start saving bandwidth today!

FreeBSD in Xen Cloud Platform (XCP)

The Xen Cloud Platform (XCP) is an open source hypervisor for creating and managing Virtual Machines (VMs). While FreeBSD is a Xen-aware operating system, there currently is no native XCP support for FreeBSD. I asked for this feature on the Citrix mailing lists and Dave Scott told me it could be possible to run FreeBSD in XCP if it met some requirements.

What you will learn…

- XCP's advantages and how to take them with FreeBSD
- How to run FreeBSD in XCP

What you should know…

- How to build the FreeBSD kernel with XENHVM stock config
- Medium notions of virtualization, Xen
- Some knowledge of debugging production environments

First, I started investigating what FreeBSD was able to do with XCP which is a Linux Xen distribution with the XAPI included and some proprietary software. I discovered that FreeBSD is able to be suspended, for example, which is key to moving a VM between hosts.

So, I started thinking that FreeBSD VM's in Hardware Assisted Virtualization (HVM) mode should be able to take advantage of suspending and moving the same way as supported Linux distributions in XCP or Windows in HVM mode using the Windows's Citrix Xenserver tools.

Some Concepts

For officially supported operating systems, Citrix provides "XenServer Tools" packages which differ depending on the guest OS and the expected mode to run in XCP. Generally, they provide programs which allow an OS to take advantage of XCP's virtualization. They could include a Xen-aware kernel in (ParaVirtualization) PV or HVM mode, XenStore management utilities, scripts for maintaining unprivileged entries in the XenStore database, or scripts which generate what is written to Xenstore.

As of FreeBSD 8.0, the GENERIC kernel on the i386 and amd64 architectures supports HVM. However, installing a custom kernel which adds the XENHVM option adds PV drivers and improves performance. This article was tested using a custom kernel running on a FreeBSD 9.0 amd64 system.

Additionally, the FreeBSD ports collection contains sysutils/xen-tools, which installs several small programs for manipulating and updating the XenStore entries and for debugging Xen related problems. These tools are needed in order to enable the XenCenter, the XCP management client, to pass the XenAPI commands to the hypervisor in order to take advantage of XCP's features.

The Configuration

1. Install the amd64 version of FreeBSD into a dom0. During installation, in the Distribution Select menu, use the arrow and spacebar to select src. This is needed to install the XENKVM kernel.
2. After booting into the FreeBSD installation, install the XENKVM kernel by typing the following commands as the superuser:

```
cd /usr/src
make buildkernel KERNCONF=XENKVM
make installkernel KERNCONF=XENKVM
```

3. Before rebooting into the custom kernel, edit `/etc/fstab` and change `ada0` to `ad0`. Next, edit `/etc/rc.conf` and change the name of the network interface from its current value to `ifconfig _ xn0`.

Once you have installed FreeBSD in HVM mode, when you right-click its entry in XenCenter, you will notice that its XCP feature set is limited. An example is seen in Figure 1.

The menu entry to "Install XenServer Tools" will not work as FreeBSD is considered as unsupported at this time. Instead, install the tools using the instructions at *http://wiki.xen.org/wiki/FreeBSD_64-bit_HVM_on_XCP*.

Once the tools are installed, you will now be able to use XenCenter to perform actions such as suspend, move, and adjust memory quantity through the balloon driver on your FreeBSD HVM machines. An example is seen in Figure 2.

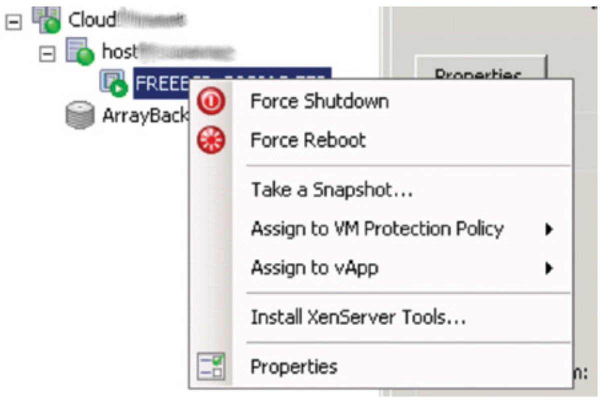


Figure 1. *FreeBSD HVM Before Installing xen-tools*

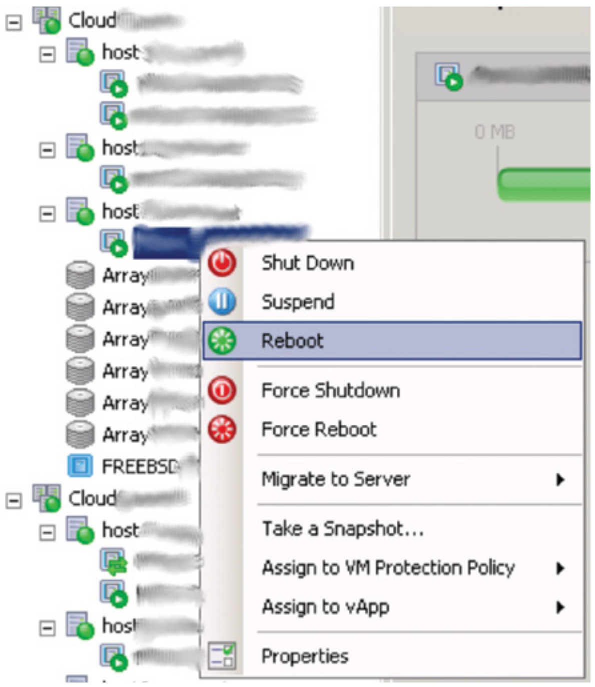


Figure 2. *FreeBSD HVM After Installing xen-tools*

Main advantages of using this virtualization environment

- The core virtualization engine is open and well-tested.
- Resize your vm on demand (in terms of memory).
- Move the machines between hosts (if you use shared storage).
- Cloning vms.
- Gradual disk usage (the amount of space configured is not reserved in full in advance).
- Optimized usage of CPU.

Special thanks to

- Dave Scott, who initially encouraged me to work on this adaptation.
- Mark Felder because of his assiduity and effort when he talks to FreeBSD and Citrix people and for maintaining the Xen-tools port.
- All of Sarenet's people, because working with them is a really nice experience.
- My family, for their unwavering support.

EGOITZ AURREKOETXEA AURRE

Egoitz Aurrekoetxea Aurre is a sysadmin and system's programmer at Sarenet, who believes that Open Source community, documentation, and software are basically the most powerful strengths in the computing world. You can reach him at: egoitz@sarenet.es; http://www.sarenet.es.
Don't hesitate to send him your comments or questions regarding his article.

VERISIGN

vBSDCon · SAVE THE DATE!
DULLES, VA · OCTOBER 25-27, 2013

Please join us October 25-27, 2013 at the Hyatt in Dulles, Virginia for the first biennial vBSDCon event. This exciting weekend will bring together members of the BSD community for a series of roundtable discussions, educational sessions, best practice conversations, and exclusive networking opportunities. See below for details on this industry weekend not to be missed:

AGENDA
- **Friday, October 25:** Evening Reception
- **Saturday, October 26:** General Session, Birds of a Feather Sessions
- **Sunday, October 27:** General Session, Breakout Sessions

WHO SHOULD ATTEND
- Developers • Engineers • Administrators • Innovators

TOPICS
- PkgNG w/ Baptiste Daroussin
- A comprehensive look at bsdinstall with Devin Teske
- Netflix Demo/Presentation with Scott Long
- netmap with Luigi Rizzo
- Migration from GCC to LLVM/Clang with David Chisnall

REGISTRATION INFORMATION WILL BE SENT TO YOU IN MAY!

Questions? Please contact: eventsteam@verisign.com

VerisignInc.com

NET OPEN SERVICES IS AN APPLICATION HOSTING COMPANY FOCUSED ON OPEN SOURCE APPLICATIONS MANAGEMENT IN HIGH AVAILABILITY ENVIRONMENT.

NET OPEN SERVICES IS PROUD TO PROVIDE A HIGH QUALITY SERVICE TO OUR CUSTOMERS SINCE 10 YEARS.

OUR EXPERTISE INCLUDES:

- **CLOUD COMPUTING, PUBLIC, PRIVATE AND HYBRID CLOUD MANAGEMENT (OPENSTACK, CLOUDSTACK, RED HAT ENTERPRISE VIRTUALIZATION)**
- **REMOTE MONITORING AND MANAGEMENT 24/7**
- **NETWORKING AND SECURITY (OPEN BSD, IP TABLE, CHECKPOINT, CISCO,...)**
- **OS AND APPLICATION MANAGEMENT (FREE BSD, OPEN BSD, SOLARIS, UNIX, LINUX, AIX, MS WINDOWS)**
- **DATABASE MANAGEMENT (ORACLE, MYSQL, CASSANDRA, NOSQL, MS SQL, SYBASE...)**
- **MANAGED HOSTING IN CARRIER CLASS DATA CENTERS**
- **DISASTER RECOVERY**

Net Open SERVICES

WE PROVIDE SERVICES IN EVERY STEP OF THE PROJECT LIFE, DESIGN, DEPLOYMENT, MANAGEMENT AND EVOLUTIONS. **NETOPENSERVICES** TEAM INCLUDES EXPERIENCED LEADERS AND ENGINEERS IN THE INTERNET SERVER INDUSTRY.

OUR TEAM HAS **15 YEARS OF EXPERIENCE** IN DEVELOPING INTERNET INFRASTRUCTURE-GRADE SOLUTIONS AND PROVISIONING INTERNET DATACENTERS AND GLOBAL SERVICE NETWORKS TOGETHER.

WE OFFER EXCEPTIONAL HARDWARE SUPPORT AS SOFTWARE SUPPORT ON UNIX/LINUX AND OPEN SOURCE APPLICATION. **NETOPENSERVICES** DELIVERS THESE CUSTOM-BUILT LINUX AND UNIX SERVERS, AS WELL AS PRECONFIGURED SERVERS AND SCALABLE STORAGE SOLUTIONS, TO OUR CUSTOMERS. WE ALSO OFFER CUSTOM DEVELOPMENT AND ADVANCED-LEVEL UNIX/LINUX CONSULTING SOLUTIONS.

WWW.NETOPENSERVICES.COM • CONTACT@NETOPENSERVICES.COM

Headquarters:
San Jose, CA

855.GREP.4.IX | Contact Us

99% Compatibility

online now...

IXSYSTEMS AND YOU ARE THE PERFECT MATCH

SHARED INTERESTS

- ✓ Enterprise Storage Solutions
- ✓ Personalized Customer Service
- ✓ Bold New Information Technology

I'm a Storage Reseller

In The EU

Looking for Storage Solutions to Sell

A Technology Partner
More Technical Experience
New Business Opportunities

Visit Today!

iXsystems

Technology Partner Seeking Resellers/Integrators for TrueNAS™ Storage Appliance

WWW.IXSYSTEMS.COM/PERFECTMATCH